

EMBEDDED SOFTWARE DEVELOPMENT WITH ECOS™

Anthony J. Massa



EMBEDDED SOFTWARE DEVELOPMENT WITH ECOS

Anthony J. Massa



PRENTICE HALL
PROFESSIONAL TECHNICAL REFERENCE
UPPER SADDLE RIVER, NJ 07458
WWW.PHPTR.COM
WWW.PHPTR.COM/MASSA/

Library of Congress Cataloging-in-Publication Data

Massa, Anthony J.

Embedded software development with eCos / Anthony J. Massa
p. cm.--(Bruce Perens' Open source series)

ISBN 0-13-035473-2

1. Embedded computer systems--Programming. 2. Application software--Development. 3. Real-time data processing. I. Title. II. Series.
QA76.6 .M364317 2002
005.26--dc21

2002035507

Editorial/production supervision: *Techne Group*
Cover design director: *Jerry Votta*
Cover design: *Anthony Gemmellaro*
Art director: *Gail Cocker-Bogusz*
Interior design: *Meg Van Arsdale*
Manufacturing buyer: *Maura Zaldivar*
Editor-in-Chief: *Mark L. Taub*
Editorial assistant: *Kate Wolf*
Marketing manager: *Bryan Gambrel*
Full-service production manager: *Anne R. Garcia*



© 2003 Pearson Education, Inc.
Publishing as Prentice Hall Professional Technical Reference
Upper Saddle River, New Jersey 07458

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.

For information regarding corporate and government bulk discounts please contact:
Corporate and Government Sales (800) 382-3419 or corpsales@pearsontechgroup.com

Other company and product names mentioned herein are the trademarks or registered trademarks of their respective owners.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-035473-2

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Educación de México, S.A. de C.V.
Pearson Education—Japan
Pearson Education Malaysia, Pte. Ltd.

*This book is dedicated to my girls,
Katie and Deanna.
You mean the world to me.
I love you.*

About Prentice Hall Professional Technical Reference

With origins reaching back to the industry's first computer science publishing program in the 1960s, Prentice Hall Professional Technical Reference (PH PTR) has developed into the leading provider of technical books in the world today. Formally launched as its own imprint in 1986, our editors now publish over 200 books annually, authored by leaders in the fields of computing, engineering, and business.

Our roots are firmly planted in the soil that gave rise to the technological revolution. Our bookshelf contains many of the industry's computing and engineering classics: *Kernighan and Ritchie's C Programming Language*, *Nemeth's UNIX System Administration Handbook*, *Horstmann's Core Java*, and *Johnson's High-Speed Digital Design*.

PH PTR acknowledges its auspicious beginnings while it looks to the future for inspiration. We continue to evolve and break new ground in publishing by providing today's professionals with tomorrow's solutions.



C O N T E N T S

Foreword	xv
Preface	xvii
Chapter 1 An Introduction to the eCos World	1
1.1 Where It All Started—Cygnus Solutions	1
1.2 The Origins of eCos	2
1.2.1 In a Word: Configurability	3
1.2.2 The eCos Configuration Method	4
1.2.3 eCos Core Components	5
1.2.4 Processor and Evaluation Platform Support	6
1.2.5 eCos Support	6
1.3 Architecture Overview	8
1.3.1 eCos Terminology	8
1.3.1.1 Component Framework	8
1.3.1.2 Component Repository	10
1.3.1.3 Configuration Options	13
1.3.1.4 Components and Packages	14
1.3.1.5 Targets	14
1.3.1.6 Templates	15
1.4 Summary	16

Chapter 2 The Hardware Abstraction Layer	17
2.1 Overview	17
2.1.1 HAL Directory Structure	19
2.1.1.1 Example HAL Function Call Trace	22
2.1.2 HAL Macro Definitions	23
2.1.3 HAL Configuration	24
2.1.3.1 Common Configuration Components	25
2.1.3.2 Architecture-Specific Configuration Components	25
2.1.4 HAL Startup	26
2.2 Summary	29
Chapter 3 Exceptions and Interrupts	31
3.1 Exceptions	31
3.1.1 HAL and Kernel Exception Handling	32
3.1.2 Application Exception Handling	38
3.2 Interrupts	40
3.2.1 eCos Interrupt Model	40
3.2.1.1 Interrupt and Scheduler Synchronization	41
3.2.2 Interrupt Configuration	42
3.2.3 Interrupt Handling	44
3.2.4 Interrupt Control	50
3.2.4.1 Interrupt Service Routine Management	51
3.2.4.2 Interrupt State Management	53
3.2.4.3 Interrupt Controller Management	54
3.3 Summary	58
Chapter 4 Virtual Vectors	59
4.1 Virtual Vectors	59
4.1.1 Virtual Vector Configuration	63
4.1.2 Virtual Vector Table Initialization	64
4.1.2.1 Communication Channels	67
4.2 Summary	71
Chapter 5 The Kernel	73
5.1 The Kernel	73
5.1.1 Kernel Directory Structure	74
5.1.2 Kernel Startup	75
5.1.3 The Scheduler	77
5.1.3.1 Multilevel Queue Scheduler	79

5.1.3.2	Bitmap Scheduler	81
5.1.3.3	Priority Levels	81
5.1.3.4	Scheduler Configuration	83
5.2	Summary	84
Chapter 6	Threads and Synchronization Mechanisms	85
6.1	Threads	85
6.1.1	Thread Stacks and Stack Sizes	94
6.2	Synchronization Mechanisms	95
6.2.1	Mutexes	95
6.2.2	Semaphores	101
6.2.3	Condition Variables	105
6.2.4	Flags	110
6.2.5	Message Boxes	113
6.2.6	Spinlocks	118
6.3	Summary	120
Chapter 7	Other eCos Architecture Components	121
7.1	Counters, Clocks, Alarms, and Timers	121
7.1.1	Counters	125
7.1.2	Clocks	129
7.1.3	Alarms	130
7.1.4	Timers	133
7.2	Asserts and Tracing	134
7.3	ISO C and Math Libraries	138
7.4	I/O Control System	140
7.4.1	I/O Sub-System	142
7.4.2	Device Drivers	146
7.5	Summary	148
Chapter 8	Additional Functionality and Third-Party Contributions	149
8.1	Compatibility Layers	150
8.1.1	POSIX	150
8.1.1.1	EL/IX	151
8.1.2	μITRON	152
8.2	ROM Monitors	152
8.2.1	CygMon	153
8.2.2	RedBoot	153
8.2.3	GDB Stub	154

8.3	File Systems	155
8.3.1	ROM File System	157
8.3.2	RAM File System	158
8.3.3	Journalling Flash File System Version 2	160
8.4	PCI Support	160
8.4.1	PCI Library API	161
8.5	USB Support	165
8.6	Networking Support	167
8.6.1	OpenBSD	168
8.6.2	FreeBSD	169
8.6.3	lwIP	170
8.6.4	Networking Threads	170
8.6.5	Networking Configuration	171
8.6.6	Networking Tests	176
8.6.7	DNS Support	178
8.7	SNMP Support	179
8.8	The GoAhead Embedded WebServer	180
8.9	Symmetric Multi-Processing Support	182
8.10	Additional Features	183
8.11	Summary	184
Chapter 9	The RedBoot ROM Monitor	185
9.1	Overview	185
9.2	RedBoot Directory Structure	187
9.3	Installation and Configuration	188
9.3.1	RedBoot Configuration	189
9.3.2	Host Configuration	193
9.3.2.1	Serial	193
9.3.2.2	Ethernet	194
9.4	User Interface and Command Set	195
9.4.1	RedBoot Commands	196
9.4.1.1	Boot Scripting	204
9.5	Summary	206
Chapter 10	The Host Development Platform	207
10.1	Overview	207
10.2	Configuring the Windows Host	209
10.2.1	Installing the Cygwin Native Tools	210

10.2.1.1	Cygwin Tools Directory Structure	217
10.2.1.2	Upgrading the Cygwin Tools	219
10.2.2	Installing the Platform-Specific Cross-Development Tools	220
10.2.3	Installing the eCos Development Kit	223
10.2.3.1	eCos Development Kit Directory Structure	229
10.2.4	Accessing the Online eCos Source Code Repository	229
10.2.4.1	Installing WinCVS	230
10.2.4.2	Setting WinCVS Preferences	235
10.2.4.3	WinCVS Update Commands	236
10.3	Summary	238
Chapter 11	The eCos Toolset	239
11.1	Packages	239
11.1.1	Package Directory Structure	240
11.1.2	The Component Definition Language Overview	243
11.1.2.1	CDL Script Files	243
11.2	The Configuration Tool	248
11.2.1	Screen Layout	248
11.2.1.1	Saving Configurations	251
11.2.1.2	Importing and Exporting Configurations	253
11.2.1.3	Configuration Window	254
11.2.1.4	Conflicts Window	255
11.2.1.5	Properties Window	256
11.2.1.6	Short Description Window	256
11.2.1.7	Output Window	256
11.2.1.8	Memory Layout Window	256
11.2.1.9	Memory Layout Manipulation	257
11.2.2	eCos Repository Database	264
11.2.3	Graphical Representation of CDL Script Files	266
11.2.4	Using Templates	270
11.2.4.1	Conflicts and Resolutions	272
11.2.5	Package Control	274
11.3	Other eCos Tools	274
11.3.1	The Package Administration Tool	275
11.3.2	The Command-Line Configuration Tool	277
11.4	Building the eCos Tools	277
11.5	Additional Open-Source Tools	277
11.5.1	Source-Navigator	278

11.5.2	Splint	279
11.6	Summary	280
Chapter 12	An Example Application Using eCos	281
12.1	The eCos Build Process	281
12.1.1	A Closer Look	282
12.2	Examples Overview	285
12.2.1	Development Hardware Setup	286
12.2.2	eCos Tools	288
12.3	RedBoot	288
12.3.1	Building RedBoot	288
12.3.2	Installing RedBoot	292
12.3.3	Booting RedBoot	293
12.4	eCos	295
12.4.1	Building eCos	295
12.5	Application	298
12.5.1	Building the Application	299
12.5.2	Loading the Application	303
12.5.3	Debugging the Application	305
12.5.3.1	Using the GDB Command-Line Interface	309
12.6	The eCos Tests	310
12.7	Simulators	311
12.8	Summary	313
Chapter 13	Porting eCos	315
13.1	Overview of Porting	315
13.2	A Platform Porting Example	317
13.2.1	PowerPC HAL Directory and File Structure	320
13.2.2	Porting Hints	334
13.3	Summary	335
Appendix A	Supported Processors and Evaluation Platforms	337
Appendix B	eCos License	345
B.1	eCos License	345
B.2	GNU General Public License	346
B.2.1	Version 2, June 1991	346
B.2.2	Preamble	346
B.2.3	How to Apply These Terms to Your New Programs	352

Appendix C	Cygwin Tools Upgrade Procedure	355
Appendix D	Building the GNU Cross-Development Tools	361
	About the Author	369
	Index	371
	About the CD-ROM	392

FOREWORD

In 1997, there were over 100 commercially supported embedded operating systems, none of which had more than a minority share of the overall embedded OS market, not to mention countless thousands of others developed for specific projects (cell phones, radar arrays, networking equipment, etc.) that had no application developer base beyond that specific project. In short, the embedded operating systems market was highly fragmented, and the cost of this fragmentation was beginning to seriously limit the viability of many embedded software projects and the OEMs who funded those software projects.

While it was clear to many that the major embedded software companies needed to change their business models in radical ways, each company believed that it could somehow outlast its competition, and that it could consolidate the market through a strategy of attrition rather than a strategy of innovation. At Cygnus Solutions, we couldn't wait for 90 percent of the market to give up; moreover, we weren't sure we wanted to serve a market that was 90-percent dead. Therefore, we took up our own challenge to create an embedded operating system that could address the incredible variety of possible embedded system designs, from the very small to the highly complex, using a single source base.

In our market research, we found two primary reasons why people wrote their own RTOSes: first, they didn't want to pay per-unit royalties to a third party, and second, they didn't want to suffer the indirect cost of code that they didn't write/control/understand using up resources within their systems. The fact that writing and debugging an RTOS is expensive (in time and money), and the fact that most custom RTOSes required a complete understanding of the entire system in order to make the smallest manual changes, often resulted in systems that were both more expensive to maintain and inferior in functionality to commercial alternatives, but such were the compromises required to avoid direct and indirect per-unit costs.

The design philosophy of eCos was to augment an open-source RTOS (which meant no per-unit royalties) with source-level configuration tools that would enable embedded developers to scale their RTOS from hundreds of bytes to hundreds of kilobytes without needing to manually change a line of source code. Of course, if some code needed to be rewritten to meet some unique requirement, open-source licensing meant the option was there. However, for most cases the 200+ configuration points supported by eCos resulted in systems that were faster to build (all the hard work was coded into the configuration rules) and resulted in smaller systems than manual methods could produce (because the automated rules were more all-seeing and all-knowing than most embedded developers could afford to be).

Since releasing eCos in 1998, we have seen it develop both a healthy user base and a strong base of talented contributors. With the publication of this book, eCos reaches a new milestone: a completely independent source of technical information about eCos, and a rather complete one at that. While this book is primarily targeted at RTOS engineers, it remains accessible to both technical managers and developers who might use, but not actually maintain, an RTOS.

The scope of the book covers the very latest information about eCos, including a section on using eCos compatibility layers to provide POSIX, μ Ittron, and even embedded Linux API compatibility with the EL/IX. Indeed, as high-end embedded system design consolidates around embedded Linux, eCos is becoming even more important for two reasons: because it provides a platform for migrating to Linux APIs without the overhead of running a full Linux-based system, and because eCos is the basis of RedBoot, the new standard ROM monitor that Red Hat supports for its embedded Linux ports.

Anthony's book is easily the most complete treatment of eCos system development. I believe it is destined to become part of every eCos developer's library.

Michael Tiemann
CTO, Red Hat, Inc.

P R E F A C E

Whether you're working on an existing project or moving on to a new development, eventually you're going to have to decide on what Real-Time Operating System (RTOS) to use. Numerous questions arise, including how much does it cost to get started, are there royalties associated with using the RTOS, what is the quality of the tools, is source code available, what features are available for the RTOS, and so on. In most situations, the lowest-cost solution in both upfront costs and royalties is the best solution, as long as it works. Eliminating royalties is very important for high-volume products, where every nickel counts.

There are also concerns of previous investments made, both in developer knowledge and financially, for the current solution. Anxiety can occur when considering moving existing code to a new software platform, which can be intimidating depending on the size of the project. Porting a new RTOS to your hardware platform can create more trepidation.

Decisions about whether to develop your own RTOS or use an off-the-shelf solution surface in some cases as well; especially when specific functionality is needed for a specialized hardware platform. In some cases, rolling your own RTOS might be the only solution. However, you can put your development way ahead by leveraging software that is already implemented, tested on numerous platforms and in various situations, and, most importantly, proven because it is successfully running on other shipping products. This eliminates the need for implementing functionality that is readily available.

This book focuses on one solution to these concerns: the Embedded Configurable Operating System (eCos). The open-source and royalty-free nature of eCos allows it to be downloaded, set up, and used, and here's the key: at no cost. When finished with this book, you will have a complete embedded software development environment—all the tools necessary to tackle any project.

Since eCos is open source, you, the developer, are in complete control over your embedded software destiny. Even the tools described in the eCos development system are open source, thereby allowing you to become completely self sufficient—although the eCos development community is out there available to lend help when needed.

Book Layout and Overview

Let's take a look at the layout of the book and get an overview of what is covered and where it is located. This enables you to focus on the specific aspects of eCos that you need to understand.

The layout of the book is intended to build on information covered in earlier chapters. We start with understanding the key components within eCos, then move to additional functionality offered in the system, and finally, get down to using eCos and the development environment.

For developers new to the eCos world, or embedded software altogether, it is helpful to understand the components that make up the eCos system by starting at the beginning. This gives the baseline understanding of the different features provided by eCos. You can then implement these software components in an actual system.

More experienced developers looking for an evaluation of eCos can skip to the later chapters and begin experimenting right away. The format of the development platform installation and examples allow a quick setup of the tools and immediate results. This lets you answer the question, "will eCos work for me?"

Current eCos users can fill in any holes that might be present in their eCos knowledge, by looking at some of the eCos concepts from a different point of view.

Chapter 1, *An Introduction to the eCos World*, begins with a brief introduction to eCos, which includes a background about the eCos open-source project and the company behind its start. A description of the eCos terminology is detailed as well. This terminology is used throughout the book and in the eCos development community. The beginning of the book is intended to provide developers who are unfamiliar with eCos a means to become acquainted with the eCos open-source project.

Next, we discuss the key components within the eCos system, presenting a closer look under the hood of these major software modules. The key component chapters offer an understanding about how the different software modules work independently and together to provide functionality required by the system.

Chapter 2, *The Hardware Abstraction Layer*, focuses on the software closest to the hardware that enables higher-level software modules to be unaware of the low-level functioning of the hardware.

In Chapter 3, *Exceptions and Interrupts*, we detail exceptions and interrupts and show how they are set up and handled in the eCos system. We discuss virtual vectors in Chapter 4, *Virtual Vectors*, which provide a means to share services between ROM and RAM applications.

The heart of the eCos RTOS, the kernel, is the focus in Chapter 5, *The Kernel*. The kernel supplies the scheduling functionality and the synchronization mechanisms for the software. Moving on to Chapter 6, *Threads and Synchronization Mechanisms*, we discuss the basic unit of

execution in eCos, the thread, and provide a detailed look at the various synchronization mechanisms supported by eCos.

Chapter 7, *Other eCos Architecture Components*, continues with our look at the different eCos components by focusing on timing components, asserts and tracing functionality, and the I/O control system.

Chapter 8, *Additional Functionality and Third-Party Contributions*, includes a broader look at some of the additional features available for eCos developed by the eCos project maintainers and third-party contributors. These include networking support, ROM monitors, file systems, PCI support, USB support, and the GoAhead WebServer.

In Chapter 9, *The RedBoot ROM Monitor*, we focus on the RedBoot ROM monitor. This standalone program is designed for embedded systems to provide a debugging and bootstrap environment. RedBoot is an eCos-based application and uses the eCos Hardware Abstraction Layer (HAL) for its foundation.

We begin our hands-on experience in Chapter 10, *The Host Development Platform*, with the installation of the host development tools. We discuss the Cygwin native tools, the GNU cross-development tools, and the eCos development kit. We also cover the installation of a Concurrent Versions System (CVS) client, WinCVS, to enable access to the online eCos source code repository. This gives you the ability to take advantage of any bug fixes or extended functionality contributed to the eCos source code.

In Chapter 11, *The eCos Toolset*, we delve into the eCos toolset with a detailed look at how the tools operate on the eCos source code, and the layout of the tools. Also included are some other open-source tools to round out and complete our open-source embedded development system. This prepares us for the next step, putting the tools to work to build our application.

Chapter 12, *An Example Application Using eCos*, lets you put your knowledge to work. The chapter starts with an overview of the eCos build process, followed by a build of the RedBoot ROM monitor. We then do a build of the eCos real-time operating system, and, finally, put it all together by building an application. This chapter provides a baseline on which you can then add additional components to assemble a system to meet your embedded software requirements.

Finally, Chapter 13, *Porting eCos*, closes with a look at porting eCos onto another hardware platform. This is key to getting your application running on your new target hardware platform, which is typically the main goal in embedded software development.

Development System and Examples

As mentioned previously, in Chapter 10 we go through the process of setting up an eCos development system. This development system includes the native Cygwin tools for Windows, the GNU cross-development tools (binutils, compiler, and debugger), the eCos configuration and management tools, a CVS client, and a lint program.

This system enables you to configure and build the eCos library, which is then linked with an application to run the eCos RTOS. The RedBoot ROM monitor is also built using this system.

After following the steps in Chapter 10, a complete open-source embedded software development environment is configured.

We go through examples of building RedBoot, the eCos library, and an application in Chapter 12. Rather than requiring a specific development board to run the examples, a second PC is needed for the target platform. This is a better approach to becoming familiar with the development tools since it's typically pretty easy to find a spare PC lying around.

Although GNU cross-development tools binary files are included on the CD-ROM for the Intel x86 and PowerPC processor architectures, the instructions for configuring and building the GNU cross-development tools for other processor architectures are provided in Appendix D, *Building the GNU Cross-Development Tools*.

The embedded software development tools are installed, and examples are built, on a Windows development system. However, the necessary files to get an embedded software development system up and running on a Linux system are included on the CD-ROM. Since the eCos configuration tools are able to run on both Linux and Windows, the procedure for building and running the examples applies to both host operating systems.

The CD-ROM accompanying this book contains the files needed to set up the complete embedded software development system for eCos as detailed in Chapter 10. The examples in Chapters 12 and 13 are also contained on the CD-ROM under the `examples` directory.

A web site is available to download the source code and updates. The site is located online at:

www.phptr.com/massa/

If you find any errors that need correction, feel free to contact me and I will update the source code accordingly.

Some Notes about the Book

All example code in this book is in C or assembly language. eCos also uses the Component Definition Language (CDL), an extension of the existing Tool Command Language (Tcl) scripting language. We cover this in Chapter 11.

In the text where a 32-bit hexadecimal value is shown, the most significant 16 bits and least significant 16 bits are separated by an underscore (“_”) (e.g., 0xABCD_EF12) for readability.

Sidebars are also included, which are used to point out important or additional information. The sidebars look like the following:

NOTE Example of a sidebar.

This book includes several Uniform Resource Locator (URL) links showing where additional information can be obtained on the Internet. The most up-to-date links are included in the text; however, we all know that links can have a finite existence.

Item lists throughout the book detail the eCos kernel API functions. These lists contain a field named “Context.” This field shows the context from where the specified function can be

called. The different contexts are Initialization, Thread, ISR, and DSR. “Any” is used to designate that the function can be called from any of the contexts.

A Brief Take on Open Source Development

There have always been “agnostics” in the debate of open source versus closed source—it sometimes comes down to whether the funding is available to purchase software tools and support. Each has its benefits and shortcomings, so let’s take a brief look at some of the pros and cons developers have found when working on both sides of the open-source and closed-source fence. What it ultimately comes down to is, does the product work, and can the schedule and cost budget be met using this solution?

Open source can be a confusing term. In an article written by Daniel Benenstien,¹ he states that open source is free software in the sense of freedom of knowledge exchange. Much more reliance is placed on the individual developer to find problems, correct the problems, and make the solution available to others in the open-source community. This iterative process is the method that creates a more robust and bug-free code base.

Presumably, because the draw of talent is worldwide, the best and brightest developers are working to make the product better. With closed source, a development support team is assembled to work on fixing problems for code they probably did not create. The team assembled might be very talented; however, the pool of talent from which to draw is very small compared to a worldwide talent pool.

Often times, closed-source, or proprietary, software and open-source software work hand in hand to complement each other. Many proprietary products are derived from open-source projects. Proprietary software must be innovative and differentiate itself from the open-source alternative; otherwise, why would people pay for something that they can get for free? Looking at things from this point of view, open source pushes proprietary software to the extreme of innovation.

There are some general advantages and disadvantages to using open-source software. One advantage is that since the source is available if you are not able to get support from the open-source community, you can dive right in and find out exactly what is going on with the code. There is no need to wait for support from an external source, which can reduce debug time drastically.

Another advantage of open-source software is that it is not tied to any one specific company. In the case of proprietary software, if the owning company changes direction, or disappears, then the application developers using that solution are left out in the cold. Open-source software prevents this because developers have full access to the code and can choose the path for their own system software.

Sometimes vendors selling proprietary software are not always the most responsive to all customers using their product. In cases where a project is not destined to produce large volumes, the vendor’s response to questions might not be as rapid as needed. The reality is that a company with higher revenue products will get the preferential treatment.

¹ Benenstien, Daniel. “Galileo Linux Multimedia Communicator.” *Embedded Linux Journal* (July/August 2001): 14–19.

Having the source also gives you the ability to implement your own changes and customize the code exactly the way you need to for your specific application. In addition, only with open-source can the source-level configuration method, as we find in eCos, be used.

Although some proprietary software vendors offer their source code to developers, there is often a fee associated with getting said code.

Security is often looked at as a negative aspect of open-source developments. Because all of the source code is available to everyone, malicious developers can exploit security holes. On the other hand, the community of developers supporting the open-source project work quickly on a fix because it is in their best interest as well. You are not at the mercy of a single source providing a fix to the security problem.

A great source for getting viewpoints from some of the leading figures behind the open-source movement is *Open Sources: Voices from the Open Source Revolution*.²

Acknowledgments

This book could not have been completed without the hard work and tremendous effort of other people. I would like to start by thanking the technical reviewers, Jonathan Larmour (a.k.a. Jifl)—I appreciate your very insightful comments—and the technical support (occasionally real-time) throughout the development of this book. Other reviewers I am indebted to for sharing their keen comments and extremely valuable views include Grant Edwards, Bart Veer, Bill Gatliff, Larry Mittag, and Paul Beskeen. I would like to thank Michael Tiemann for writing the Foreword for this book.

I would like to thank the editor, Mark Taub—I appreciate the feedback and support you gave throughout this process.

Many thanks to the companies and open-source projects whose software is included on this CD-ROM—some brilliant stuff there.

Closing on a Personal Note

I would like to thank my Nonno and Nonna for all their support throughout my life. They are always there for me providing whatever is needed whether it's encouragement, a getaway to the backcountry for a ride and lunch, or a trip to the Boll Weevil. I sure miss Nonna. I love you both very much.

I would like to thank my brother, Laurie, and sister, Catherine, for their encouragement and understanding. You two are the best brother and sister anyone can ask for. By the way, I'd also like to congratulate my sister on passing the CPA exam, although I do believe the score is now 2 to 1—in my favor. :) I love you guys.

Thank you Mom and Dad for your never-ending support and encouragement for things I attempt in my career, and especially for things I attempt throughout my life. You are always there for me in whatever I do. I am very thankful to have such wonderful parents. I love you with all my heart.

² DiBona, Chris; Ockman, Sam; Stone, Mark. *Open Sources: Voices from the Open Source Revolution* (O'Reilly, 1999).

I would like to thank my wonderful daughter, Katie. You always could sense when I needed to take a break while working on this book. You not only knew that I should take a break, but you insisted that I leave the office immediately and forced me to watch one of your shows—although my preference is *Seinfeld*. Thanks for that; it really helped me to clear my head and refocus. You are very special to me and I love you with all my heart.

And, last but certainly not least, a big thank you to my wife, Deanna. Well, it's all over now. Thank you for supporting me on this effort and all efforts I undertake for us. Thank you for giving me the time to work on the book. I know it was difficult at times and put a lot of responsibility on you, but I hope the journey was worth it. Thank you not only for being a wonderful wife, but also for being my best friend. I love you, always.

Finally, I hope you all enjoy this book.

Anthony J. Massa
amassa@san.rr.com

An Introduction to the eCos World

In this first chapter, we take a brief look at the origins of the Embedded Configurable Operating System (eCos) and the people and company behind it. We then get an overview of the configurable architecture of eCos, the core functionality, the different processors and evaluation platforms supported, and technical assistance options available.

Lastly, we get an overview of the eCos architecture and a look at the terminology used to describe the different pieces of the configuration system. The overview gives us a general idea of the components we detail in later chapters, and the terminology described in this chapter is used throughout the book.

1.1 Where It All Started—Cygnus Solutions

Michael Tiemann, David Henkel-Wallace, and John Gilmore founded Cygnus Solutions in 1989. The idea behind Cygnus Solutions was to provide high-quality support and development for open source software. It was initially unclear whether this business model would work out; however, by the end of the first year it was obvious from the value of the support and development contracts that the business was real. The workload was enormous for the five-person company (the three founders, a salesperson, and a part-time graduate student).

It was clear that the engineering support model worked; however, the costs to fulfill these contracts were very high. In order to generate income at a lower cost, the engineers had to put their heads together to come up with an idea. The plan was to focus their development efforts on a small set of open-source technology that could be sold. The key to maintaining this development on an order that could be handled by the group was to keep the focus very small. What they came up with was selling the GNU compiler (GCC) and debugger (GDB) as shrink-wrapped software. This was the right team of people to do the job. Michael Tiemann, who contributed

numerous GNU compiler ports and also wrote the first native C++ compiler (GNU C++ or G++), took on the task of working on GCC; David Henkel-Wallace worked on the binary utilities (binutils) and the library; and John Gilmore worked on GDB.

This task grew to monumental proportions. One advantage, or so it seemed, was that John Gilmore decided to become the new GDB maintainer. Making this known to the Internet community immediately flooded him with different versions of GDB. Now came the task of integrating these new version features.

Eventually, the hard work paid off in what today is called the GNUPro Developers Kit. The kit includes:

- **GCC**—the highly optimized ANSI-C compiler.
- **G++**—ANSI-tracking C++ compiler.
- **GDB**—source- and assembly-level debugger.
- **GAS**—GNU assembler.
- **LD**—GNU linker.
- **Cygwin**—UNIX environment for Windows.
- **Insight**—a graphical user interface (GUI) for GDB.
- **Source-Navigator**—source code comprehension tool.

1.2 The Origins of eCos

Initial design discussions for eCos began in the spring of 1997. The primary goal was to bring a cost-effective, high-quality embedded software solution to the marketplace. This new development would also complement the existing GNUPro tools, thereby expanding Cygnus' product offering.

Another essential requirement was that eCos needed to be designed in such a way that a small resource footprint could be constructed. By working with different semiconductor companies, Cygnus was able to architect a real-time operating system (RTOS) that abstracted the hardware layer and was highly configurable. This enabled the RTOS to fit into many diverse embedded systems. The highly configurable nature of eCos also allowed companies to reduce time to market for embedded products.

Reducing cost is always a concern in embedded systems. By using the open-source model, eCos was available with no initial costs. It could be downloaded and “test driven” free of charge. In addition to eliminating startup costs, another attractive cost-saving feature was that eCos had no backend charges—it had to be royalty-free.

Developers have full access to the entire software source code, including the tools, which can be modified as necessary (see Appendix B, *eCos License*, for the eCos license). There are no up-front license fees for the eCos run-time source code or any of the associated tools; everything needed to set up a complete embedded software development environment can be accomplished

for free. Developers do not have to contribute back any additional components or applications developed; however, they are required to contribute back modifications to the eCos code itself. These contributions help the open-source community develop a better product.

Today, numerous companies are using eCos, and many successful products have been launched running eCos, including the Brother HL-2400 CeN network color laser printer, Delphi Communiport, and the Iomega Hip Zip Digital Audio Player.

1.2.1 In a Word: Configurability

In order to get an understanding of the eCos architecture, it is important to appreciate the component framework that makes up the eCos system. This component framework is specifically targeted at embedded systems and meeting the requirements associated in embedded design. Using this framework, an enormous amount of functionality for an application can be built from reusable software components or software building blocks. The eCos component framework has been designed to control components to minimize memory use, allow users to control timing behavior to meet real-time requirements, and use usual programming languages (e.g., C, C++, and assembly for certain implementations in the Hardware Abstraction Layer [HAL]).

Most embedded software today provides more functionality than what might actually be needed for a particular application. Often, extra code is included in a software system that gives generic support for functionality that embedded developers are not concerned with and is not needed. This extra code makes the software unnecessarily more complex. Furthermore, the more code, the greater the chance of something going wrong. An example would be a simple “Hello World” program. With most RTOSes, full support for mutexes, task switching, and other features would be included, even though it is not necessary for such a simplistic application. eCos gives the developer ultimate control over run-time components where functionality that is not needed can easily be removed. eCos can be scaled from a few hundred bytes up to hundreds of kilobytes when features such as networking stacks are included and third-party contributions such as Web servers are used.

Developers are able to select components that satisfy basic application needs, and configure that particular component for the specific implementation requirements for the application. This could mean enabling or disabling a particular feature within a component, or selecting a particular implementation for the component. An example of this is in the kernel scheduler configuration. eCos offers the developer options such as the ability to select the number of priority levels and whether time slicing is used. Any code unnecessary to meeting the developer’s requirements is eliminated in the final image of the application.

Configurability allows a company to build an internal foundation of reusable components with access to the source code of the component. This can reduce development time and time to market because the components are highly portable and can be used in a wide range of applications. The eCos framework encourages third-party development to extend the features and functionality of the core eCos components. As more and more developers work toward extending the functionality on products and contribute these components back to the eCos project, the growth

in functionality of eCos is limitless. Moreover, if the functionality is presently not available, the source code is there to accomplish the task yourself.

1.2.2 The eCos Configuration Method

As embedded systems are pushed to be smaller, faster, cheaper, and more sophisticated, control over all software in the system is necessary. There are different methods to control the behavior of components included in an application image. The philosophy of the eCos component control implementation is to reduce the size for systems that have resource limitations, even to the detriment of systems that do not have strict resource constraints. With this design philosophy in mind, minimal systems do not suffer from additional code necessary to support advanced features only used in more complex systems.

One method to control software components is at run time. In this method, no up-front configuration of the component is done. The code linked to the application provides support for all behaviors of the component whether it is required by the application or not, causing the code size to be much larger. An example of run-time control is an application that runs on a desktop. When the application is executed from the disk drive, the shared libraries (Dynamic Link Libraries [DLL], etc.) needed by the application are loaded when the application starts.

Another method for component control is at link time. In this case, the code can use only the specific functions of a component that it needs, and the code that supports functionality not needed by the application is left out. Many linkers, such as the GNU linker (ld), offer link-time control, or commonly called, *selective linking*. With selective linking, unreferenced functions and data are removed from the application image. However, this is still insufficient because only entire functions can be removed—an all-or-nothing approach.

Compile-time control gives the developer control of the component behavior at the earliest stage, allowing the implementation of the component itself to be built for the specific application for which it is intended. Compile-time control gives the best results in terms of code size because the control is at the individual statement level in the source code rather than at the function or object level. This makes compile-time control very well suited for embedded development.

eCos uses compile-time control methods for its software components, along with selective linking provided by the GNU linker. Using compile-time control or source-level configuration is achieved by using the C preprocessor. An example of source-level configuration is shown in Code Listing 1.1. The flag `INCLUDE_FUNCTIONALITY` is either enabled or disabled by the developer. When this section of the code is compiled, only the code that is needed is included in the application image.

```
1  #ifdef INCLUDE_FUNCTIONALITY
2
3  ...
4
5  #else
```

```
6
7  ...
8
9  #endif
```

Code Listing 1.1 Example code of source-level configuration.

With source-level configuration, very specific options can be applied in the code, which is appropriate for embedded systems since the majority of embedded applications compile into static images.

In addition to generating smaller code, source-level configuration offers many other advantages important in embedded software development:

- Applications are faster because variables do not have to be checked during run time to determine what action to take.
- The code is more responsive and latencies are reduced, which aids in creating a more deterministic system that is important in real-time devices.
- A simpler code base is generated, making verification and testing easier.
- The code is tailored for the application, creating an application-specific RTOS.
- Costs can be reduced because resource usage is optimized and processor cycles are efficiently used, thereby enabling less expensive hardware to be specified in the design.

The Configuration Tool, provided with the eCos release, eases the selection and configuration of the software components. The tool also provides the capability to build the eCos framework from the software building blocks selected, which are then linked with the application. The Configuration Tool runs on both Windows and Linux platforms. A detailed look at the Configuration Tool is presented in Chapter 11, *The eCos Toolset*.

1.2.3 eCos Core Components

Certain standard functionality is expected in a real-time embedded operating system, including interrupt handling, exception and fault handling, thread synchronization, scheduling, timers, and device drivers. eCos delivers these standard components with the real-time kernel as the central core. The core components are:

- **Hardware Abstraction Layer (HAL)**—providing a software layer that gives general access to the hardware.
- **Kernel**—including interrupt and exception handling, thread and synchronization support, a choice of scheduler implementations, timers, counters, and alarms.
- **ISO C and math libraries**—standard compatibility with function calls.
- **Device drivers**—including standard serial, Ethernet, Flash ROM, and others.
- **GNU debugger (GDB) support**—provides target software for communicating with a GDB host enabling application debugging.

Both eCos and the application run in supervisor mode. In the eCos system, there is no division between user and kernel mode.

A minimal test infrastructure is included with eCos. The tests are configured in a similar way to the application, which ensures that the exact configuration selected is tested. The Configuration Tool provides the facilities for administering the tests. Expanding the current test infrastructure is planned in future eCos releases.

1.2.4 Processor and Evaluation Platform Support

eCos supports a wide variety of popular embedded processor architectures. This makes eCos a great choice for companies using many diverse hardware architectures on different product lines. Once the eCos HAL has been ported to a new architecture, the application layer can be moved over seamlessly to support the new application requirements.

The eCos software support is for standard commercial evaluation platforms on the market today. The main processor architectures supported include:

- ARM
- Fujitsu FR-V
- Hitachi H8/300
- Intel x86
- Matsushita AM3x
- MIPS
- NEC V8xx
- PowerPC
- Samsung CalmRISC16/32
- SPARC
- SPARClite
- SuperH

Appendix A, *Supported Processors and Evaluation Platforms*, lists the specific processor ports and evaluation platforms supported by eCos. Since many ports are contributed back to the eCos project for the benefit of others to use, the eCos Web site, at <http://sources.redhat.com/ecos/hardware.html>, is the best source for finding the most recent contributions for the eCos project.

1.2.5 eCos Support

Getting support is always a concern when trying to determine whether a certain product should be used. There are different means for getting support with eCos. The route used for obtaining support will greatly depend on the amount of assistance needed.

There are six different mailing lists available for the eCos project:

- **Discussion List**—contains support and technical assistance on various topics about the eCos project from developers. The list can be found online at <http://sources.redhat.com/ml/ecos-discuss>, and the email address for the list is ecos-discuss@sources.redhat.com.
- **Patches List**—used for submitting eCos patches for approval by the maintainers before they are committed to the source code repository. The list also hosts discussions about the different patches submitted. This list can be found online at <http://sources.redhat.com/ml/ecos-patches>, and the email address for posts is ecos-patches@sources.redhat.com.
- **Development List**—includes discussions about current enhancements being developed, such as new ports and new features. General requests for help and information about eCos should be kept to the discussion list. This list can be found online at <http://sources.redhat.com/ml/ecos-devel>, and the email address is ecos-devel@sources.redhat.com.
- **Announcement List**—a low-volume list for significant news about eCos that is also used to announce new eCos releases or major feature enhancements. This list can be found online at <http://sources.redhat.com/ml/ecos-announce>, and the email address is ecos-announce@sources.redhat.com.
- **CVS Web Pages List**—contains notifications of changes to the eCos Web pages that are maintained in Concurrent Versions System (CVS). This read-only list can be found online at <http://sources.redhat.com/ml/ecos-webpages-cvs>.
- **CVS List**—a read-only list that gives notifications of changes made to the eCos source code repository. This list can be found online at <http://sources.redhat.com/ml/ecos-cvs>.

As with all mailing lists, it is generally a good idea to dig in and try to find the answer to your problem before posting a previously reported, and solved, question to the list. To assist with this process, the eCos mailing lists provide a means for searching previous messages posted to the list.

Through my development using eCos, I have found that the discussion list is very responsive, compared to facilities provided by other companies in the past. I have found a two to three-day turnaround on getting answers to questions I have posted to the list from developers in the eCos community. However, it should be understood that not all questions asked on the discussion list are answered. Having questions that are very specific and detailed often helps the maintainers, and other developers, to lend support.

A great advantage of open-source development, and an open discussion list, is that the community for the project is there to assist in answering questions. Quite often, users who have encountered similar problems will post answers to aid their developer colleagues.

Users are able to subscribe to any of the lists and receive emails for all messages posted to a particular list or a digest form that contains a collection of messages from the particular list. Receiving individual messages can be somewhat overwhelming; therefore, subscription to the digest version of the discussion list might be better. To sign up for any of the mailing lists, you can go online at:

<http://sources.redhat.com/ecos/intouch.html>

Bug tracking for the eCos project is contained in a Bugzilla database. The Bugzilla database has an advanced search engine that allows searches based on keywords, for particular platforms, and specific versions of eCos. The Bugzilla database search engine can be found online at:

<http://bugzilla.redhat.com/bugzilla/query.cgi?product=Red%20Hat%20eCos>

Additional information about other features about the Bugzilla bug tracking software, including a new bug report form, can be found online at:

<http://bugzilla.redhat.com/bugzilla>

If there are private technical issues to discuss about eCos development and collaboration, the maintainers have an email address to reach them directly at:

ecos-maintainers@redhat.com

1.3 Architecture Overview

eCos is designed as a configurable component architecture consisting of several key software components such as the kernel and the HAL. The fundamental goal is to allow construction of a complete embedded system from these reusable software components. This allows you to select different configuration options within the software component, or remove unused components altogether, in order to create a system that specifically matches the requirements of your application. By creating an eCos image that closely matches your system requirements, the size of the software is compact, only including used components. The software application is also faster because extra code is not executed, compared to other real-time operating systems that do not offer configurability and, therefore, incorporate all functionality regardless if it is required by the application.

Figure 1.1 shows an example of how the core building blocks, and some of the optional components available in the eCos system, can be layered together to incorporate the functionality needed for a specific application.

Since configuration is a key aspect of the eCos system, tools are provided to manage the complexity of the different configuration options. These tools also allow components to be added or removed as needed. The tools build the main end product of an eCos configuration, which is a library that can be linked with application code.

1.3.1 eCos Terminology

The eCos configuration system involves some key terms that are important to understand. These terms are used in eCos documentation and throughout this book.

1.3.1.1 Component Framework

The collection of tools that allow users to configure the eCos system and manage different packages in the repository is called the *component framework*. Included in the component framework are the command-line configuration tool, the graphical Configuration Tool, the Memory Layout

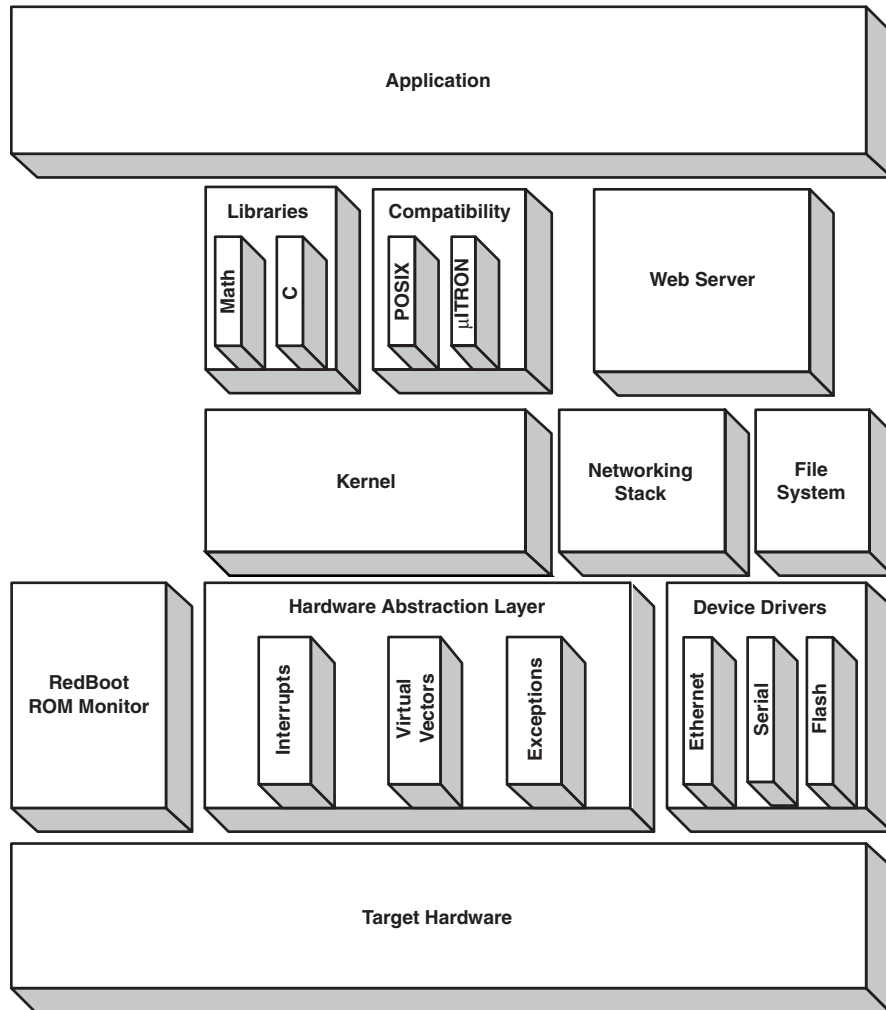


Figure 1.1 Example embedded software system showing layering of eCos packages.

Tool, and the Package Administration Tool. How these tools are used to manage and build an eCos configuration image is detailed in Chapter 11.

The component framework saves the collection of choices into a *configuration*. A configuration contains the packages that have been selected, as well as the status of options within the package describing whether the option is enabled, disabled, or set to a particular value. The framework tools operate on the configuration as a whole using the *properties* of configuration options to determine things such as default values and valid option ranges. The configuration is saved in a file with a `.ecc` extension. The relationship between a configuration and the values in the `.ecc` file is described in Chapter 11.

Figure 1.2 shows a portion of the *eCos Kernel* package from the Configuration Tool. The figure shows how the building blocks are encapsulated within each other to create a complete and independent package. We can see the hierarchy of the configuration from packages to components to configuration options to suboptions. Building blocks are grouped together in a package based on the functionality they include. In Figure 1.2, we see the *eCos Kernel* package, which contains the *Kernel Exception Handling* component and the *Kernel Schedulers* component; the other eCos Kernel components are not shown in this figure. We can see in Figure 1.2 the nesting of configuration options, such as *Scheduler Timeslicing*, and suboptions that compose the components. The different modules, components, and options are described further later in this section. Additional information about the Configuration Tool can be found in Chapter 11.

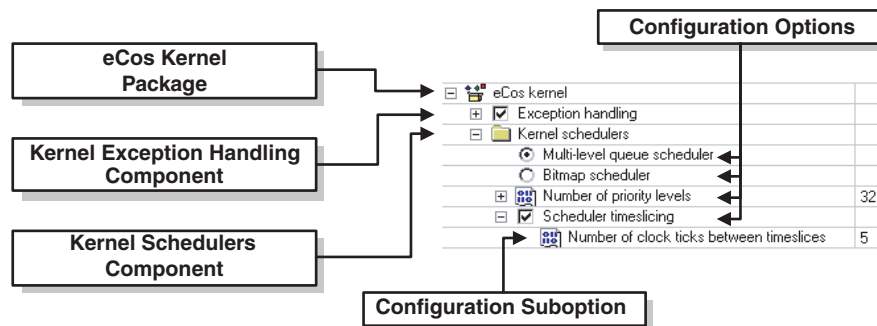


Figure 1.2 Example of the configuration building blocks that compose a package.

1.3.1.2 Component Repository

The *component repository* is a directory structure containing all packages from an eCos installation. The component framework includes a Package Administration Tool for adding new packages, updating current packages, and removing old packages within the repository. The main directory, *ecos*, contains the eCos distribution files. The subdirectory that contains the component repository is *packages*. A database file, *ecos.db* (located in the *packages* directory), is maintained by the Package Administration Tool and contains the details about the various packages in the component repository.

Occasionally, the database file needs to be hand edited. For example, when porting a HAL to your own hardware platform, editing the database file allows the new HAL to be recognized and controlled by the configuration tools. We will go through the process for editing the database file in Chapter 13, *Porting eCos*. In general, application developers can treat the repository as a read-only resource that can be reused for different applications. Figure 1.3 gives a high-level overview of the component repository directory structure.

Because eCos is an evolving code base with new contributions available all the time, the directory structure shown in Figure 1.3 is a snapshot of the eCos version 2 component repository. It is intended to show the overall layout of the eCos source code components rather than

specifics about the directories. When new contributions to the eCos project are made, the maintainers decide if the contribution belongs under an existing subdirectory or requires the start of a new subdirectory. The latest eCos repository can be found online at:

<http://sources.redhat.com/cgi-bin/cvsweb.cgi/?cvsroot=ecos>

The details about configuring a system to use the latest source code found in the online repository are covered in Chapter 10, *The Host Development Platform*.

A description of the component repository directory structure is given in Table 1.1. Details of the directory structure and file contents for packages can be found in Chapter 11.

Table 1.1 Component Repository Directory Structure Descriptions

Directory	Description
compat	Contains packages for the POSIX (IEEE 1003.1) and μ ITRON 3.0 compatibility.
cygmon	Package contents for Cygmon standalone debug monitor. ^a
devs	Includes all device driver hardware-specific components such as serial, Ethernet, and PCMCIA.
error	Contains common error and status code packages. This allows commonality among packages for error and status reporting.
fs	Includes the ROM and RAM file system packages.
hal	Incorporates all HAL target hardware packages.
infra	Contains the eCos infrastructure such as common types, macros, tracing, assertions, and startup options.
io	Packages for all generic hardware-independent Input/Output (I/O) system support, such as Ethernet, flash, and serial, which is the basis for system device drivers.
isoinfra	Contains package that provides support for ISO C libraries (such as stdlib and stdio) and POSIX implementations.
kernel	Includes the package that provides the core functionality (such as the scheduler, semaphores, and threads) of the eCos kernel.
language	Incorporates the packages for the ISO C and math libraries, which allows the application to use well-known standard C library functions and the floating-point mathematical library.

Table 1.1 Component Repository Directory Structure Descriptions (*Continued*)

Directory	Description
net	Packages for basic networking support including TCP, UDP and IP, and the SNMP protocol and agent support libraries based on the UCD-SNMP project.
redboot	Contains package for the RedBoot standalone debug ROM monitor.
services	Includes packages for dynamic memory allocation and support for compression and decompression library.

^a The RedBoot ROM monitor has replaced the Cygmon debug monitor.

1.3.1.3 Configuration Options

The *configuration option* is the fundamental unit of configurability in the eCos system. Typically, a configuration option corresponds to a single choice you can make. This choice might be to enable, disable, or to set a value for the option. Configuration options have a macro associated with them. The macro is used in the source-level configuration control. Each macro has a sensible default value that can be used as a baseline. Once the application is built and running, the options can be tuned to meet the specific requirements of the system. The configuration options selected can affect which files are built into the eCos library, or cause certain values to be set in a particular file. In turn, selection of certain configuration options allows you to have control down to a particular source code line in some circumstances.

The component framework uses a Component Definition Language (CDL) to describe the package. Within each package is at least one CDL script file. This script file describes the package to the component framework. Detailed information about the CDL can be found in Chapter 11.

The configuration options detailed in this section are text names used by the graphical Configuration Tool. At this time, the configuration option and the relationship with its associated CDL name are unimportant. Throughout the book, the CDL names for specific components or options are given as reference.

The *nesting* of configuration options is used to give finer control over the system. This nesting of configuration options is shown in Figure 1.2. The configuration option *Scheduler Timeslicing* contains the configuration suboption *Number of Clock Ticks Between Time Slices*. If *Scheduler Timeslicing* is enabled, a value, in this case 5, for the suboption can then be selected. If *Scheduler Timeslicing* is disabled, the suboption setting is irrelevant and cannot be set within the Configuration Tool.

A particular configuration option might have dependencies on other options in the configuration. These dependencies, or *constraints*, are sometimes straightforward where one configuration option requires that another option be enabled. For example, in Figure 1.2, selecting the *Bitmap Scheduler* configuration option requires that *Scheduler Timeslicing* be disabled.

Other times, configuration options cannot be modified. Take the case of processor endianness. Some processors are hard-wired to operate in a specific endian mode, and others can be programmed to operate in either big-endian or little-endian mode at runtime. Depending on the hardware selected, endianness might not be a configuration option that can be modified. In other configuration options, the constraint might be a range for a particular value. For example, the configuration option *Number Of Priority Levels* has a constraint range of 1 to 32, which is currently set to 32 in Figure 1.2. Specifying a value out of this range is not allowed in the Configuration Tool.

As configuration options are modified, *conflicts* might arise because certain constraints are not satisfied. The configuration tools report these conflicts allowing us to take corrective action. These conflicts can be bypassed; however, compile-time or link-time failure might occur. Conflicts should be resolved before continuing with the system configuration. The configuration tools try to resolve conflicts that arise during the configuration process. The tools might apply a solution automatically or ask us for intervention in solving the conflict. Additional information about conflicts can be found in Chapter 11.

1.3.1.4 Components and Packages

A *component* is a configuration option that encapsulates more detailed options within it. Entire components can be enabled or disabled, depending on the needs of a particular application. For example, in Figure 1.2, the *Kernel Exception Handling* component can be disabled by unchecking the box next to the component. Disabling the component causes all configuration options under that component, as well as any files associated with the component, to be irrelevant and not included in the build. This hierarchy of encapsulation gives us control of the configuration at a higher level. Eliminating unused components also reduces the compile time of the eCos image.

Another example where component control is useful is in the case where a particular device on the target hardware, such as an Ethernet port, is not going to be used in the application. Eliminating the device driver component for the Ethernet port reduces memory usage in the system.

A *package* is a type of component that is ready for distribution. Incorporated in a package are all necessary source code files, header files, configuration description files, documentation, and other relevant files. A package is often contained in a single file, allowing it to be installed with the appropriate tool or updated in the future when changes are made. Having a distribution package as a standalone unit allows third-party developers to extend the functionality offered in the eCos system. Enabling a package loads the configuration data into the appropriate tool. You also have control over the version of the packages that are used in the system.

1.3.1.5 Targets

A *target* is the piece of hardware on which the application will be executed. The target might be an off-the-shelf evaluation board, your own hardware platform, or a simulator. When creating a configuration, you select a target so that the component framework can load particular packages to support the devices and HAL relevant to the target. In addition, configuration options are changed from their default values to settings appropriate for the target.

The process is more automated for evaluation boards supported by eCos, whereas using your own hardware requires more involvement to determine what packages are to be loaded and the value of configuration option settings.

1.3.1.6 Templates

A *template* is a partial configuration that gives us a valid starting point. Templates are a combination of a hardware target and a group of packages. The group of packages is given a name, as shown in Table 1.2, to describe the functionality included. eCos comes with a small number of default templates. When a new configuration is created, a template is used as a starting point to match the general needs of the application. Configuration options can then be fine-tuned to meet more specific requirements you have. The configuration tools show the specific packages included in the template.

Table 1.2 eCos Templates

Template Name	Description
All	Provides all packages for a particular hardware target.
Cygmon	Includes packages necessary to build eCos with Cygmon.
Cygmon_No_Kernel	Incorporates packages for building Cygmon without eCos kernel support.
Default	Contains the infrastructure, kernel, C and math libraries, plus necessary support packages.
Elix	Provides packages for supporting EL/IX compatibility.
Kernel	Includes the HAL, infrastructure, and eCos kernel packages.
Minimal	Incorporates the HAL and infrastructure packages only.
Net	Contains necessary support packages for using the OpenBSD networking stack.
New_Net	Provides support packages for using the FreeBSD networking stack.
Posix	Provides HAL, infrastructure, eCos kernel, and POSIX packages.
RedBoot	Used for building the RedBoot ROM monitor image.

Table 1.2 eCos Templates *(Continued)*

Template Name	Description
Stubs	Includes packages necessary to build eCos GDB stubs.
Uitron	Provides full level S (standard) compliance with version 3.02 of the μ ITRON standard, plus many level-E (extended) features.

1.4 Summary

This chapter gave us a brief background of eCos and the company behind it. We looked at the compile-time or source-level configuration eCos uses and the advantages it brings to embedded applications. Next, we examined the different mailing lists available for getting support with eCos.

Finally, we went through the eCos terminology used in this book and eCos documentation, which gives us a baseline of the elements that compose the eCos system. We are now ready to take an in-depth look at the eCos system, the software components available, and how we use eCos.

The Hardware Abstraction Layer

This is the first of eight chapters that describe the eCos architecture and its components. The eight eCos architecture chapters provide an explanation of the core software components, which are the building blocks for the eCos system. Many of these building blocks are common to several real-time operating systems; however, we need to understand how these components operate and interact with each other in the eCos system. This will prepare us for the next part of the book, which enables us to set up and configure the eCos tools to build an image for use in your applications.

In this chapter, we get into the details of the *Hardware Abstraction Layer* (HAL). It is particularly important to understand the architecture of the HAL, because when the time comes to port eCos onto your own hardware, the HAL is the software component that will need to be adapted to support the new hardware platform.

In Chapter 3, we cover exceptions and interrupts. Chapter 4 describes virtual vectors. In Chapter 5, we look at the core of the eCos system: the kernel. Chapter 6 details threads and synchronization mechanisms. Chapter 7 covers the other eCos components such as counters, timers, libraries, and the I/O control system. Next, Chapter 8 describes other functionality and contributions available for eCos, such as networking support, file systems, and PCI support. Finally, we conclude the eCos architecture with Chapter 9, which details the RedBoot ROM Monitor.

2.1 Overview

The HAL isolates architectural-dependent features and presents them in a general form to allow portability of other infrastructure components. Basically, the HAL is a software layer, with generalized Application Programming Interfaces (API), which encapsulates the specific hardware operations needed to complete the desired function.

An example that demonstrates how the HAL abstracts hardware-specific implementations for the same API call is shown in Code Listing 2.1 for the ARM architecture, and in Code Listing 2.2 for the PowerPC architecture.

```

1  #define HAL_ENABLE_INTERRUPTS()      \
2      asm volatile (                  \
3          "mrs r3,cpsr;"              \
4          "bic r3,r3,#0xC0;"          \
5          "msr cpsr,r3"               \
6          :                            \
7          :                            \
8          : "r3"                       \
9      );

```

Code Listing 2.1 ARM architecture implementation of HAL_ENABLE_INTERRUPTS() macro.

```

1  #define HAL_ENABLE_INTERRUPTS()      \
2      CYG_MACRO_START                 \
3      cyg_uint32 tmp1, tmp2;          \
4      asm volatile (                  \
5          "mfmsr %0;"                 \
6          "ori %1,%1,0x8000;"          \
7          "rlwimi %0,%1,0,16,16;"      \
8          "mtmsr %0;"                 \
9          : "=r" (tmp1), "=r" (tmp2); \
10     CYG_MACRO_END

```

Code Listing 2.2 PowerPC architecture implementation of HAL_ENABLE_INTERRUPTS() macro.

In Code Listings 2.1 and 2.2, we see that the call HAL_ENABLE_INTERRUPTS(), as shown on line 1 of both listings, is the same regardless of the architecture. However, the process for actually executing an interrupt enable varies from architecture to architecture, as shown on lines 2 through 9 in Code Listing 2.1 and on lines 2 through 10 in Code Listing 2.2. The HAL allows the application layer to directly access hardware and any architectural features and does not assume it is the only controller of all hardware in the system.

General design principles were followed during the architecting of the HAL. First, the entire HAL is implemented in C and assembly language. This allows the HAL to have the widest range of applicability.

Second, interfaces to the HAL are implemented in C macros. This allows the most efficient implementation to be used and yet the interface is not affected. The interfaces can be implemented as inline assembly code, inline C code, or external function calls to C or assembler code. By using the inline approach, the run-time overhead associated with a function call is eliminated; however, the size of the code can grow.

Finally, an emphasis on the ease of platform porting was made because the developers themselves typically perform this task.

The HAL consists of three separate modules (or submodules); however, the boundary between each module is intentionally fuzzy:

- Architecture
- Platform
- Variant

The first HAL submodule defines the *architecture*. Each processor family supported by eCos is considered a different architecture. Each architecture submodule contains the code necessary for CPU startup, interrupt delivery, context switching, and other functionality specific to the instruction set architecture of the associated processor family.

A second HAL submodule defines the *variant*. A variant is a specific processor within the processor family described by the architecture. An example of a feature that might be included at this level is support for an on-chip peripheral such as a Memory Management Unit (MMU).

The third HAL submodule defines the *platform*. A platform is a specific piece of hardware that includes the selected processor architecture and, possibly, a variant. This module typically includes code for platform startup, chip select configuration, interrupt controllers, and timer devices.

2.1.1 HAL Directory Structure

All HAL packages included in the repository are found under the `hal` subdirectory. Figure 2.1 shows a snapshot of the HAL directory structure for eCos version 2. At this point, the architectures included in the directory structure are not important; however, it is important to get an overview of where files containing certain HAL functionality are located within the repository. It is also important to understand that not all HAL architectures follow the same directory structure. We discuss the details about specific files within the directory structure in Chapter 11, *The eCos Toolset*.

The subdirectories under the HAL are broken down by processor architecture. As seen in Figure 2.1, the architecture subdirectories include:

- `arm`
- `calmrisc16` (for the Samsung CalmRISC16)
- `calmrisc32` (for the Samsung CalmRISC32)
- `frv` (for the Fujitsu FR-V)
- `h8300` (for the Hitachi H8/300)
- `i386` (for the Intel x86)
- `mips`
- `mn10300` (for the Matsushita AM3x)
- `powerpc`
- `sh` (for the Hitachi SuperH)

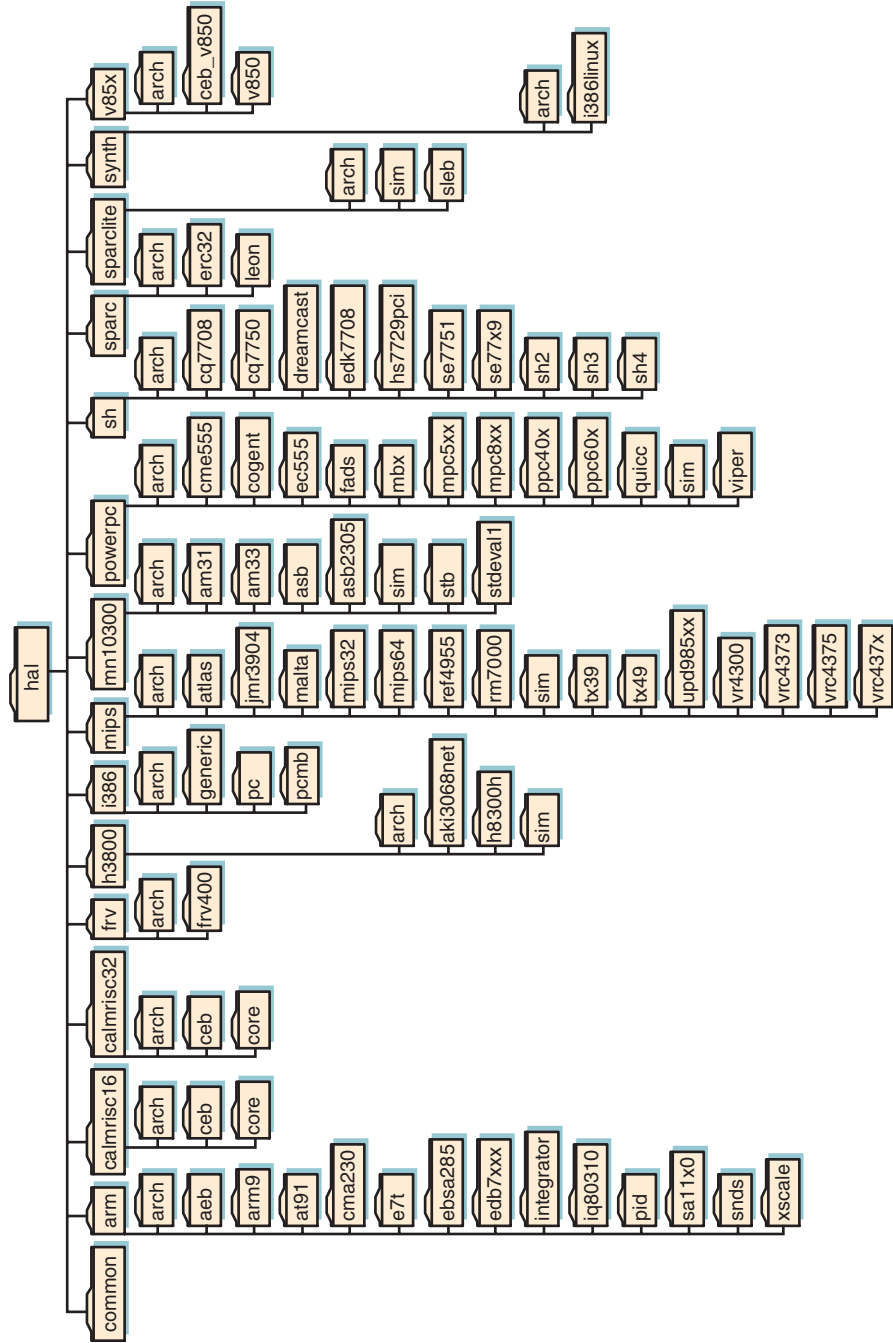


Figure 2.1 HAL directory structure snapshot.

- `sparc`
- `sparclite`
- `synth` (for the i386 Linux kernel)
- `v85x` (for the NEC V8xx)

Each architecture subdirectory includes the platform and variant support related to that particular processor.

For example, under the `powerpc` subdirectory is the `mbx` subdirectory that contains the platform package for the Motorola PowerPC MBX860 development board support. Included in the MBX package subdirectory is the code needed for platform-specific initialization such as the memory layout files, clock configuration, and chip select programming. In addition, under the `powerpc` subdirectory is the `mpc8xx` subdirectory, which contains files necessary for the different series of MPC8xx variants (including the MPC823, MPC850, and MPC860). The MPC8xx variants of the PowerPC contain code for MMU and interrupt control.

As new platform and architecture ports are developed, the package contents are inserted in the appropriate place in the HAL directory structure. Since new ports are made available at various times, the directory structure can change often to accommodate new additions.

A few subdirectories, and a description of their contents, within the HAL structure are worth noting. First is the subdirectory `common`, located under the main HAL directory. This subdirectory contains the package configuration files general to all HAL architectures, including files for general interrupt configuration, virtual vector layout, and HAL debugging control. Function wrappers are contained in this subdirectory to create the commonality found among all HAL implementations.

Another subdirectory to notice is `arch`, located under every architecture tree. The `arch` subdirectory contains files for generic support for the processor architecture. Functionality included in this generic support consists of exception vector initialization, ROM and RAM startup configuration, common interrupt and exception handling, thread context switch handling, a generic linker script file, and common debugging functions.

Some HAL architectures include a subdirectory to contain the variant code. This subdirectory is named `var`. An example of an architecture that contains this subdirectory is the ARM SA11x0.

Last is the `sim` subdirectory, which can be found under the architecture trees that support processor simulators. The architecture simulators provide a simple model of the processor rather than detailed operation of a particular evaluation board. When using a simulator, it is impossible to use any of the device drivers. The simulators are best used as interim targets when you need to start testing application functionality. The simulators can keep the software development task on schedule while an evaluation board or your own hardware target is under development. The architectures that have simulators are:

- Hitachi H8/300
- MIPS

- Matsushita AM3x
- PowerPC
- SPARClite

2.1.1.1 Example HAL Function Call Trace

To get a better understanding of the relationship of the submodules in the HAL and how the different functionality is split among the directory structure, let us trace a function call into the HAL. The function call we will trace is the `__reset()` function defined in the `common` subdirectory of the HAL. For this example, we will use the MIPS Atlas Evaluation board as the target hardware platform. The submodules that implement the functionality of the reset function can vary for different HAL architectures. Figure 2.2 is a graphical representation of the `__reset()` function call trace.

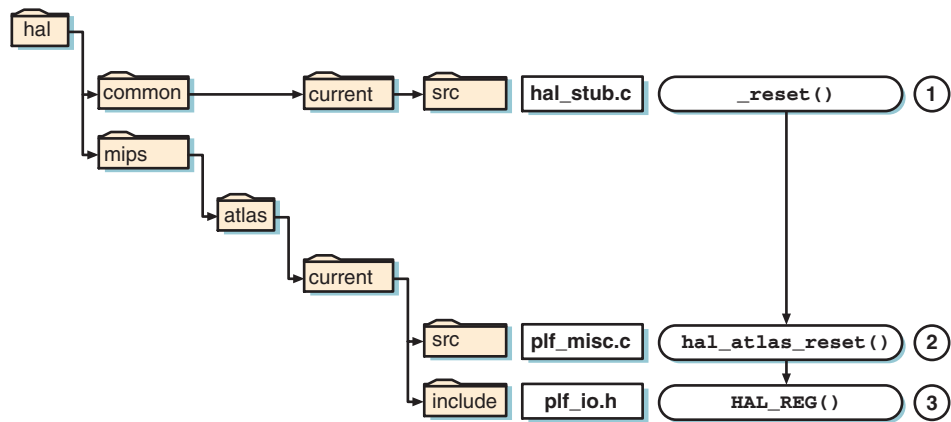


Figure 2.2 HAL reset function call trace.

A description for each of the steps numbered in Figure 2.2 follows.

1. The `__reset()` routine is the generically defined reset function for all HAL packages. The source code for this routine is found in the `hal_stub.c` file under the `common\current\src` subdirectory.
2. Next, the `hal_atlas_reset()` routine, defined in `plf_misc.c` under the `mips\atlas\current\src` subdirectory, is executed for the MIPS Atlas platform.
3. Finally, the platform reset routine uses the architecture macros defined in `plf_io.h` under the `mips\atlas\current\include` subdirectory to toggle the appropriate register within the processor. The `HAL_REG()` macro causes the write to the MIPS Atlas reset register.

The execution trace of this reset call shows us how the implementation is split from the generic definition of the reset function common to all HAL architectures—to the platform-specific reset code—and finally, to the specific processor register manipulation code.

2.1.2 HAL Macro Definitions

The HAL defines architecture macros that make a common API for encapsulating the processor-specific implementation functionality. The HAL macros are used to control the interrupt, cache, memory management, I/O, diagnostics, debugging, and architectural features for the processor, given that the processor provides the functionality. This section gives a general overview of these HAL macros and their location within the HAL architectures. Additional detailed descriptions of the HAL macros containing specific functionality (e.g., exception, interrupt, and clock macros) are described in their associated chapters.

Generally, HAL architecture macros are enclosed within `#ifndef` statements to allow the macro to be overridden in the platform or variant submodules. Code Listing 2.3 shows an example of this.

```
1  #ifndef CYGHWL_HAL_INTERRUPT_VECTORS_DEFINED
2
3      #define CYGNUM_HAL_INTERRUPT_0          0
4      #define CYGNUM_HAL_INTERRUPT_1          1
5      #define CYGNUM_HAL_INTERRUPT_2          2
6      #define CYGNUM_HAL_INTERRUPT_3          3
7      #define CYGNUM_HAL_INTERRUPT_4          4
8      .
9      .
10     .
11 #endif
```

Code Listing 2.3 Example HAL interrupt vector macro definitions.

In Code Listing 2.3 we see part of the interrupt vector definitions; in this case, for the MIPS processor architecture. Line 1 checks for the definition of the macro `CYGHWR_HAL_INTERRUPT_VECTORS_DEFINED`. The variant or platform-specific code within the HAL can override the interrupt vector definitions provided by the architecture submodule, shown on lines 3 through 7, by simply defining `CYGHWR_HAL_INTERRUPT_VECTORS_DEFINED`. The variant or platform submodule can then define the interrupt vectors appropriately.

The location of the HAL architecture code is under the `arch` subdirectory, as we see in Figure 2.1. Typically, the macros are located in `.h` files under the `include` subdirectory; however, the specific location of this functionality might differ for different HAL architectures. Table 2.1 lists some of the general HAL architecture macro filenames and describes the functionality included in the file.

Table 2.1 HAL Architecture Macro Descriptions

Filename	Description
<code>hal_arch.h</code>	Abstracts the architecture-specific functionality that includes macros for breakpoint support, thread control, and stack control. The <code>HAL_SavedRegisters</code> structure is also defined in this file. This structure defines the processor-specific registers to store the machine state. These registers are stored during context switches, exception handling, and interrupt handling.
<code>hal_cache.h</code> ^a	Provides instruction and data cache control macros, such as size definitions, synchronization, enable/disable, lock/unlock, and flushing.
<code>hal_intr.h</code>	Contains the interrupt and clock support macros. The interrupt macros include interrupt vector definitions, exception vector definitions, enable/disable control, attach/detach control, and mask/unmask control. The clock macros include control for initialization, reset, and reading.
<code>hal_io.h</code>	Includes the I/O register reading and writing macros. For example, <code>HAL_READ_XXX</code> and <code>HAL_WRITE_XXX</code> , where <code>XXX</code> defines the size of the read or write operation.
<code>hal_mem.h</code> or <code>hal_mmu.h</code> ^a	Contains the macros for defining and controlling the MMU.
<code>xxx-stub.h</code> or <code>xxx_stub.h</code> , where <code>xxx</code> defines the architecture; for example, <code>mips-stub.h</code> .	Provides the definition and control macros for GDB support. This includes functionality for getting trap information, get/set register contents, setting the program counter, single stepping, and various breakpoint controls.

^a Might not be supported by all HAL architectures.

2.1.3 HAL Configuration

As mentioned in Chapter 1, eCos uses source-level configuration control, which determines the software components included in a particular eCos image. Source-level configuration sets values for specific macros based on options you select. Then, the HAL is built according to the specifications set in the configuration.

The HAL configuration options can be split into two different parts, common and architecture-specific components. The common configuration components contain general options for most or all HAL packages within the eCos system. The architecture-specific configuration components can be further broken down into general architecture options and platform-specific options, which are relevant to a particular hardware target.

2.1.3.1 Common Configuration Components

The common configuration components are standard across all HAL packages. There are six components included in the HAL common configuration. Each component contains configuration options for setting up the HAL to meet the needs of a specific application. Item List 2.1 gives a description of the six components. The CDL component name is also shown in the list for reference. More information about the CDL and how it is used within a package can be found in Chapter 11.

Item List 2.1 HAL Common Configuration Components

Component Name **Platform-Independent HAL Options**
CDL Name CYGPKG_HAL_COMMON
Description Controls the general interfacing to the kernel and other broad options, including HAL exception support, MMU table installation, and diagnostic output routing control.

Component Name **HAL Interrupt Handling**
CDL Name CYGPKG_HAL_COMMON_INTERRUPTS
Description Allows overall configuration of the interrupt structure, such as using separate interrupt stacks maintained by the HAL, whether nested interrupts are enabled, and interrupt stack size configuration.

Component Name **HAL Context Switch Support**
CDL Name CYGPKG_HAL_COMMON_CONTEXT
Description Enables context switch code to exploit the calling conventions for a specific architecture to reduce the amount of state information saved during a context switch.

Component Name **Cache Startup Behavior**
CDL Name CYGPKG_HAL_CACHE_CONTROL
Description Allows data and instruction cache enabling during the startup process. If additional platform-specific cache configuration is needed, these options should be disabled.

Component Name **Source-Level Debug Support**
CDL Name CYGPKG_HAL_DEBUG
Description Determines the level of debug support included in the HAL. This allows the GDB debug support to be provided by a ROM monitor or contained in the HAL build itself.

Component Name **ROM Monitor Support**
CDL Name CYGPKG_HAL_ROM_MONITOR
Description Defines the interaction between the application and a ROM monitor. The application can either be built to work with a ROM monitor or behave as a ROM monitor. This determines the initialization process for exceptions, interrupts, and virtual vectors between the ROM monitor and the application.

2.1.3.2 Architecture-Specific Configuration Components

The architecture-specific components can vary greatly from platform to platform. The architecture-specific components present for configuration are dependent on the template hardware selected.

For example, using the graphical configuration tool and selecting the *Motorola MBX860/821 board* hardware template enables the following packages to be enabled for configuration (CDL package names are in parentheses):

- **PowerPC Architecture** (CYGPKG_HAL_POWERPC)
- **PowerPC MPC8xx Variant HAL** (CYGPKG_HAL_POWERPC_MPC8xx)
- **Motorola MBX PowerPC Evaluation Board** (CYGPKG_HAL_POWERPC_MBX)
- **Motorola MBX PowerQUICC Support** (CYGPKG_HAL_QUICC)

Some configuration suboptions for this hardware template include the development board clock speed selection and the ROM boot device to use.

In another case, selecting the *ARM PID Development Board* as the hardware template enables the following packages (CDL package names are in parentheses):

- **ARM Architecture** (CYGPKG_HAL_ARM)
- **ARM PID Evaluation Board** (CYGPKG_HAL_ARM_PID)

Configuration suboptions such as Thumb instruction set enabling, processor endian mode selection, and hardware diagnostic port control are available under the ARM architecture components.

One configuration suboption present for all architecture-specific components is the *Startup Type* (CYG_HAL_STARTUP). The Startup Type imposes constraints on the *ROM Monitor Support* common configuration component, and vice versa, which might cause conflicts when configuring these configuration suboptions. The Startup Type can be either ROM or RAM, and for some platforms ROMRAM—where the code is stored in ROM but copied to RAM at startup for execution.

HAL configurations with ROM startup selected must be self-contained, meaning that all initialization of the hardware is performed by the HAL contained in the application. Two general development scenarios use a ROM Startup Type. In the first, the eCos library is built for use within a ROM monitor, allowing applications to be loaded into RAM for debug. The other scenario is typically used after the application has been debugged and is ready for release.

HAL configurations with RAM startup selected typically assume the existence of a debug environment or ROM monitor. In this startup configuration, the application can rely on the ROM monitor to provide support for various interrupt and exception handling processes. You can find more information about the RedBoot ROM Monitor and how it uses the HAL in Chapter 9, *The RedBoot ROM Monitor*.

2.1.4 HAL Startup

To get a better understanding of the functionality provided by the HAL, we need to take a look at the startup process the software goes through to initialize the hardware. The different submodules of the HAL take care of different aspects of the initialization process, such as coordinating

operation with a ROM monitor, invoking static and C++ constructors, and jumping to the start of the application code.

Figure 2.3 is a flowchart of the routines involved during the initialization of the HAL for the PowerPC-based Motorola MBX860 development board. The startup procedure might differ slightly depending on the architecture and platform used, as far as when certain initialization steps are completed and the name of the routine that accomplishes the initialization task. In addition, note that the startup procedure might also deviate from what is shown in the flowchart depending on the configuration options selected for the HAL. The routines described in Figure 2.3 are implemented in either assembly language or C.

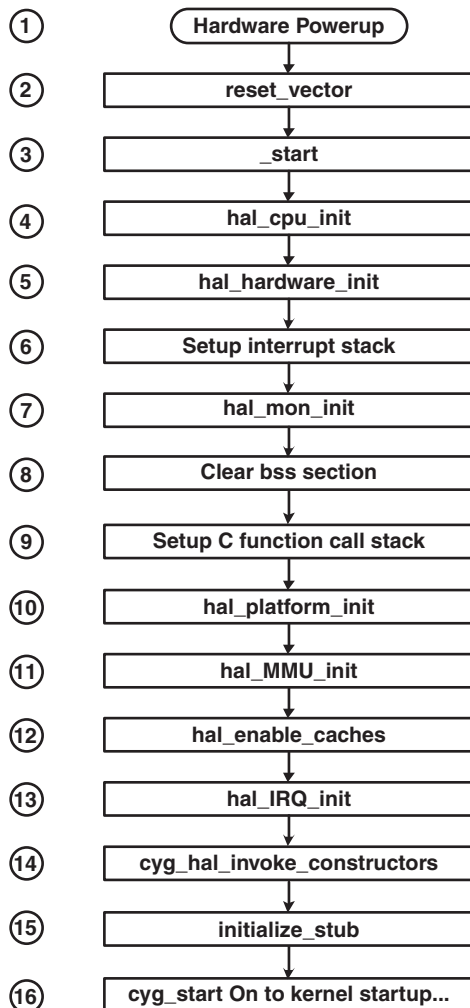


Figure 2.3 HAL startup procedure.

A description of each step numbered in the HAL startup process shown in Figure 2.3 follows:

1. The starting point for the system startup is after a power cycle has occurred, labeled `Hardware Powerup`. This startup process also applies for a soft reset startup.
2. After a hard or soft reset occurs, the processor jumps to its reset vector (cleverly called `reset_vector` in the diagram). The reset vector is found in the file `vectors.S` under the `arch` subdirectory for each HAL architecture. This file contains the starting point for all HAL packages. The reset vector performs the minimum processor register configuration to allow the system to continue with the initialization process.
3. Next, the reset vector jumps to `_start`. This is also found in `vectors.S` and is the main starting point for all HAL initialization.
4. Next, the routine `hal_cpu_init` is called, which is located in either `variant.inc` or `arch.inc` depending on the architecture. This function handles setting processor-specific registers, such as disabling instruction and data caches, to ensure that the processor is in a known state for the remainder of the initialization process.
5. The next routine called is `hal_hardware_init`. The functionality contained in this routine is platform specific and therefore found in the platform assembly file (for the Motorola MBX board, this is the `mbx.S` file). Hardware setup in this routine includes cache configuration, setting interrupt registers to a default state, disabling the processor watchdog, setting real-time clock registers, and configuring chip select registers based on the platform-specific hardware.
6. The next step is to set up an interrupt stack area. This reserves a storage area for saving processor state information when an interrupt occurs. The amount of space to reserve is configurable in the common configuration component. The startup context temporarily uses the interrupt stack to perform its initialization; for example, to make calls into C routines. Since interrupts are not enabled during this startup procedure, this does not create a conflict.
7. The code executed in the `hal_mon_init` function, which is located in the file `variant.inc` or `platform.inc`, is configuration dependent. When executing as a ROM monitor or ROM application, the main task for this routine is to ensure that default exception handlers are installed for every exception condition supported by the processor. You can find more information on exception vector configuration in Chapter 3, *Exceptions and Interrupts*.
8. The next step in the HAL initialization process is to clear the BSS section, which contains all noninitialized local and global variables with static storage class.
9. The stack is then set up so that C function calls can be made from within the `vectors.S` assembly code.
10. Next, the `hal_platform_init` routine is called, located in the `hal_aux.c` file, for a specific platform. This, in turn, calls `hal_if_init`, found in the file `hal_if.c` of the HAL common subdirectory. `hal_if_init` initializes the virtual vector table based

- on the configuration options selected. See Chapter 4, *Virtual Vectors*, for detailed information on the virtual vector table initialization and how it is used within the eCos system.
11. Initialization of the MMU, which handles translations of logical addresses to physical addresses also providing protection and caching mechanisms, is handled in the routine `hal_MMU_init` located in the file `hal_misc.c`. This file is under the `arch` subdirectory.
 12. The next step is to enable the data and instruction caches. This is done in `hal_enable_caches`, which can be found in the file `hal_misc.c` under the `arch` subdirectory for the given processor.
 13. Now, the routine `hal_IRQ_init` is executed in order to set up the Communications Processor Module (CPM), which accepts and prioritizes internal and external interrupts. This is specific to the PowerPC processor and is located in the file `hal_intr.c` under the `arch` subdirectory.
 14. Next, all global C++ constructors are called from `cyg_hal_invoke_constructors`. This routine is in the file `hal_misc.c` under the `arch` subdirectory. The linker handles the generation of the list of global constructors. The file `cyg_type.h`, under the `infra` subdirectory, contains macros that define the order in which constructors are called.
 15. If the configuration is set up for a debug environment and a ROM monitor is not providing debug support, the next routine called is `initialize_stub`, located in the HAL common subdirectory in the file `generic_stub.c`. `initialize_stub`, which installs the standard trap handlers and initializes the hardware for debug.
 16. Finally, the last step in the HAL initialization process is to turn control over to the kernel for its initialization. The routine `cyg_start` is the place for the HAL-to-kernel transition. We discuss the kernel initialization process in Chapter 5, *The Kernel*.

2.2 Summary

In this chapter, we focused on the HAL, which gives our application a generalized API to the underlying hardware. We looked at the HAL directory structure, the macros supplied by the HAL, and the configuration of the HAL. Finally, we went through the startup procedure of the HAL, allowing us to see how a target platform is initialized.

Exceptions and Interrupts

In this chapter, we begin with a look at exceptions and how they are handled at the HAL and application layers within the eCos system. Next, we cover the eCos interrupt model, including interrupt configuration, handling, and control. The exception and interrupt information prepares us for managing software and hardware events that occur.

3.1 Exceptions

An *exception* is a synchronous event that occurs during the execution of a thread that disrupts the normal flow of instructions. If exceptions are not properly processed during program execution, severe consequences, such as system failures, can occur. Exception handling is extremely important, especially in embedded systems, to avoid these failures, and improves the robustness of the software. Properly implemented exception handling can also aid in software execution recovery so that an application can proceed with its task after an exception has occurred.

Examples of exceptions that can occur in a system include those raised by hardware (such as a memory access error) and those raised by software (such as a divide by zero error). After the exception causes an interruption, the processor will jump to a defined address (or exception vector) and begin to run the instructions at that location. The address that the processor jumps to contains the exception handling code to process the error. Different processor architectures might locate the exception handlers at various places and implement this jump process in a variety of methods.

The method for implementing exception handling can vary. eCos does not provide an implementation similar to the throw and catch facilities provided in C++. For an embedded system, the simplest and most flexible method for handling exceptions is to call a function. This function needs a context or an area to do its work, and is typically passed in information, such as

the exception number and possibly some optional parameters, to process the exception. After returning from this exception handler function, the thread can continue its execution.

There are two main methods for exception handling in eCos. The first is a combination of *HAL and Kernel Exception Handling*. The HAL provides the general hardware-level exception processing and then passes control on to the application for any extended exception support needed. This is the default configuration method.

The second exception handling method is *Application Exception Handling*. This allows the application to take full control over any or all of the exceptions and attaches a vector service routine directly to the hardware. The exception handler routine needs to be written in assembly language when using this configuration method.

The configuration option names described are found in the graphical configuration tool. In parentheses are the CDL names for the given option. For example, using the graphical configuration tool you could look up the configuration option name *HAL Exception Support*, which is located in the configuration window. This configuration option has a CDL macro associated with it—in this case, `CYGPKG_HAL_EXCEPTIONS`—which is located in the Properties window of the graphical configuration tool. We cover the graphical configuration tool in Chapter 11, *The eCos Toolset*.

The configuration options that control the exception handling are the *HAL Exception Support* (`CYGPKG_HAL_EXCEPTIONS`) option, located under the *Platform-Independent HAL Options* (`CYGPKG_HAL_COMMON`) component, and the *Exception Handling* (`CYGPKG_KERNEL_EXCEPTIONS`) component under the *eCos Kernel* package. Each of these configuration components requires the other to be enabled. Enabling these options allows HAL and Kernel Exception Handling. If both of these configuration options are disabled, it is up to you to provide exception handlers for Application Exception Handling.

Figure 3.1 shows the exception handling execution flow for the two main configuration options. The gray boxes illustrate, using a system call exception as an example, the HAL and Kernel Exception Handling execution flow. The white boxes show the execution flow for Application Exception Handling using an alignment exception as an example. We look at the execution flow shown in Figure 3.1 later in this chapter.

As another option, all exception handling can be disabled in the system. If this configuration is selected and an exception occurs, the macro `CYG_FAIL` is called and an assertion is raised. If assertions are not enabled, the state of the processor is restored; however, the cause of the exception might still exist. This is not a very elegant method for exception handling and can lead to undefined system behavior when an exception occurs. However, if code space is extremely important, it might be necessary to eliminate any extended exception processing.

3.1.1 HAL and Kernel Exception Handling

It is important that an exception handler is installed for each exception supported by the processor. If a handler is not installed for a particular exception and it occurs during execution, the processor will jump to the exception address to begin execution only to find no code to execute. This can cause erratic behavior that can be very difficult to debug and might lead to system failures.

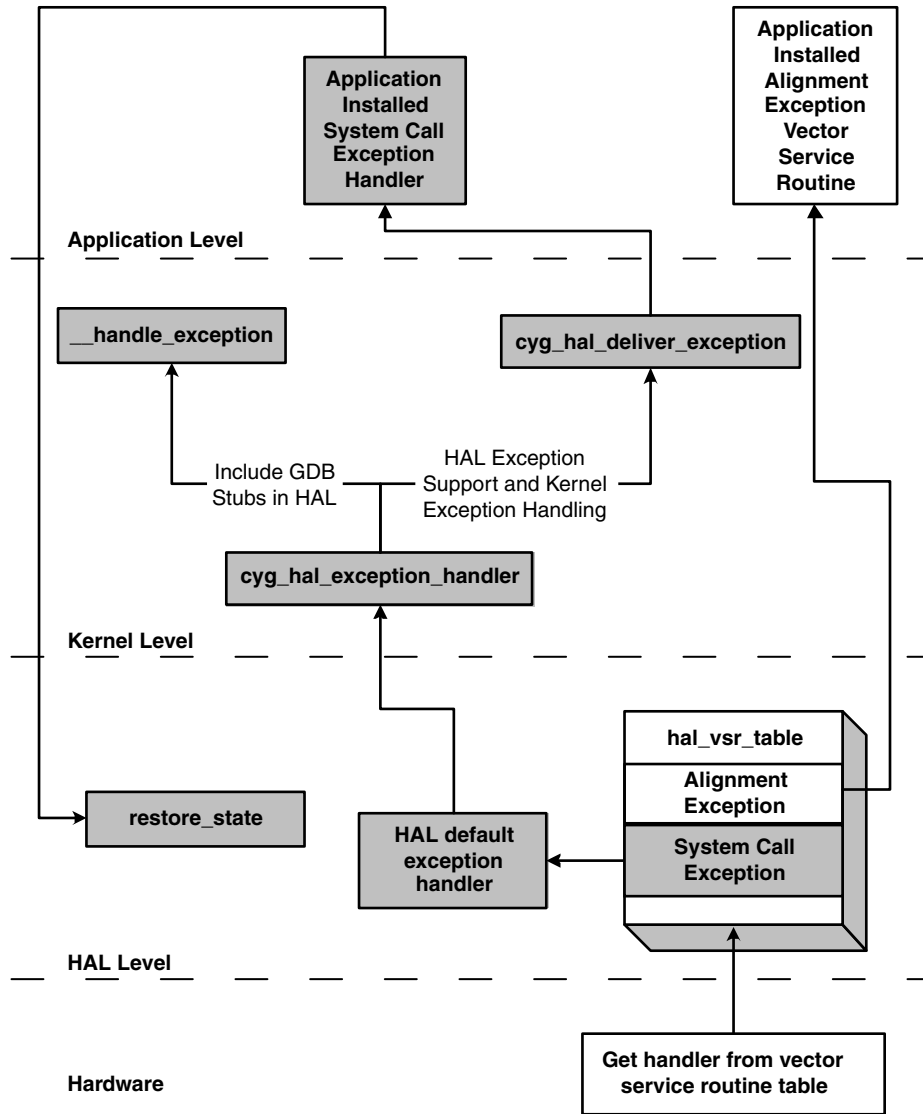


Figure 3.1 eCos exception handling execution flow.

The eCos HAL uses a *Vector Service Routine (VSR)* table that is defined in each HAL package as `hal_vsr_table`. The VSR table is an array of pointers to the exception handler routines. The VSR table is the first place the processor looks to determine where to jump to execute the exception handler.¹ The size and base address of the `hal_vsr_table` is architecture

¹ Certain HAL processor architectures, such as the i386, perform decoding of the exception vector number in order to determine where to look in the VSR table.

specific. For example, the PowerPC architecture has the option of locating the VSR table at either address `0x0000_0000` or `0xFFFF0_0000`, based on the configuration option settings of certain processor registers. The MIPS architecture specifies that the exception vector location be at address `0xBFC0_0000`. Linker script files are used to define the location of the exception vector table. Linker script files are located under the `arch` subdirectory for a given HAL architecture and have a `.ld` extension.

The VSR table is located at a fixed memory location. This allows an application running from RAM, which is a typical debug configuration, to take control over certain exception service routines while keeping the ROM monitor in charge of debug exception handlers.

The eCos HAL ensures that a default exception VSR is installed during the HAL startup process for each exception supported by a given architecture. Installation of the default exception VSR takes place in the routine `hal_mon_init`. There is one default exception VSR defined in each HAL package. Different architectures have different names for the default exception VSR, as shown in Table 3.1.

The ARM architecture is not listed in Table 3.1 because it does not use the `hal_vsr_table`. Instead, the ARM architecture defines separate handler routines for each exception it supports. These routines can be found in the file `vectors.S`.

Table 3.1 Default Exception Vector Service Routines

Architecture	Default Exception Vector Service Routine Name
CalmRISC16 ^a	<code>__default_swi_vsr</code> and <code>__default_trq_vsr</code>
Fujitsu FR-V	<code>_exception</code>
Hitachi H8/300	<code>__default_trap_vsr</code>
i386 (including Synth), PowerPC, SuperH	<code>cyg_hal_default_exception_vsr</code>
CalmRISC32, MIPS, MN10300	<code>__default_exception_vsr</code>
V8x	<code>do_exception</code>
SPARC, SPARClike	<code>hal_default_exception_vsr</code>

^a The CalmRISC16 processor contains two different default exception routines, one for a swi exception and one for a trace exception.

The job of the default exception VSR is to perform the common processing of all exceptions, which includes saving the processor's state, calling any kernel-level handler routine to perform

additional processing, and restoring the state of the processor prior to returning to normal program execution.

After the HAL has completed the common exception processing, control is passed to the kernel level. The routine that is called to handle this HAL-to-kernel transition is `cyg_hal_exception_handler`. This routine is found in the file `hal_misc.c` under the HAL arch subdirectory. The `cyg_hal_exception_handler` routine has two different execution paths possible, based on configuration option settings. If we look at Figure 3.1, we can follow the execution path for a system call exception, shown in the gray boxes. At the kernel level, the two possible configuration options are labeled on the lines leading from the `cyg_hal_exception_handler` routine.

The first kernel-level configuration option we see in Figure 3.1 is *Include GDB Stubs in HAL* (`CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS`), which is under the *Source-Level Debug Support HAL* (`CYGPKG_HAL_DEBUG`) common component. This option is typically used in a HAL package for a ROM monitor build to allow the debug stub code to process the exception. This exception processing occurs in the routine `__handle_exception`. The `__handle_exception` routine manages all debug exception processing such as breakpoints, single stepping, and debug packet protocol communication. Additional information about GDB and the GDB protocol can be found online at:

http://sources.redhat.com/gdb/onlinedocs/gdb_toc.html

The second kernel-level configuration option we see in Figure 3.1 is *HAL Exception Support* (`CYGPKG_HAL_EXCEPTIONS`), under the *General Platform-Independent HAL* common component. This HAL option requires the Kernel Exception Handling to be enabled in the eCos Kernel package. In this configuration, an application can install its own handler for exceptions to take care of any further processing, such as displaying or logging an error message.

Within the kernel *Exception Handling* (`CYGPKG_KERNEL_EXCEPTIONS`) option is the configuration suboption *Use Global Exception Handlers* (`CYGSEM_KERNEL_EXCEPTIONS_GLOBAL`), which allows installation of handlers to be either global—one set of handlers for all exceptions in the entire system—or on a per-thread basis. The default configuration for this suboption setting is to enable global exception handlers. In this configuration, the HAL exception handler calls the routine `cyg_hal_deliver_exception`, which is located in the file `except.cxx` under the kernel package subdirectory.

Item List 3.1 Kernel Exception Handler API Functions

```
Syntax: void
        cyg_exception_set_handler(
            cyg_code_t exception_number,
            cyg_exception_handler_t *new_handler,
            cyg_addrword_t new_data,
            cyg_exception_handler_t **old_handler,
            void **old_data
        );
```

Context: ²	Thread
Parameters:	<p><code>exception_number</code>—HAL architecture-specific exception definition number. Each HAL defines, in the file <code>hal_intr.h</code>, all exceptions supported starting from 0.</p> <p><code>new_handler</code>—address of application exception handler routine to be called after HAL has completed its exception processing.</p> <p><code>new_data</code>—data to be passed into exception handler.</p> <p><code>old_handler</code>—address of previously installed exception handler, or <code>NULL</code> if this is the first handler installed for the exception.</p> <p><code>old_data</code>—data of previously installed exception handler, or <code>NULL</code> if this is the first handler installed for the exception.</p>
Description:	Replace the current exception handler either globally or on a per thread basis, depending on the kernel exception handling configuration.
Syntax:	<pre>void cyg_exception_clear_handler(cyg_code_t exception_number,);</pre>
Context:	Thread
Parameters:	<p><code>exception_number</code>—HAL architecture-specific exception definition number. Each HAL defines, in the file <code>hal_intr.h</code>, all exceptions supported starting from 0.</p>
Description:	Restores the default handler.
Syntax:	<pre>void cyg_exception_call_handler(cyg_handle_t thread, cyg_code_t exception_number, cyg_addrword_t exception_info);</pre>
Context:	Thread
Parameters:	<p><code>thread</code>—current handle for the executing thread allowing the thread to call the global or thread exception handler.</p> <p><code>exception_number</code>—HAL architecture-specific exception definition number. Each HAL defines, in the file <code>hal_intr.h</code>, all exceptions supported starting from 0.</p> <p><code>exception_info</code>—this parameter is the third parameter passed into the exception handler routine.</p>
Description:	Invokes the installed exception handler for the given exception number. The return value is <code>void</code> for this function.

To control the exception service routines from the application, the eCos kernel defines functions within its API. Item List 3.1 shows the details of these functions.

```
1 #include <cyg/kernel/kapi.h>
2 #include <cyg/infra/diag.h>
3 #include <cyg/hal/hal_intr.h>
```

² If global exception handlers are used, by enabling the *Use Global Exception Handlers* configuration option, the `set` and `clear` functions can also be called during initialization.


```
4
5 //
6 // New System Call exception handler.
7 //
8 void system_call_exception_handler(
9     cyg_addrword_t data,
10    cyg_code_t number,
11    cyg_addrword_t info)
12 {
13     diag_printf( "ERROR: System Call Exception\n" );
14 }
15
16
17 //
18 // Main starting point for the application.
19 //
20 void cyg_user_start(
21     void)
22 {
23     cyg_exception_handler_t *old_handler;
24     cyg_addrword_t old_data;
25
26     //
27     // Install the exception handler for error output.
28     //
29     cyg_exception_set_handler(
30         CYGNUM_HAL_VECTOR_SYSTEM_CALL,
31         &system_call_exception_handler,
32         0,
33         &old_handler,
34         &old_data);
35 }
```

Code Listing 3.1 Example using the eCos kernel API for installing an exception handler within an application.

In Code Listing 3.1, we see an example of an application setting up the routine `system_call_exception_handler`, shown on line 8, for the PowerPC system call exception, `CYGNUM_HAL_VECTOR_SYSTEM_CALL`. `CYGNUM_HAL_VECTOR_SYSTEM_CALL` is defined in the PowerPC HAL package in the file `hal_intr.h`, which is included on line 3.

In this example, the exception handler simply writes out a message to the diagnostic port when this particular exception occurs, as shown on line 13. The function used is `diag_printf`, which is defined in the file `diag.h` included on line 2. When the call is made to install the handler, line 29, it can apply to either the thread or globally, depending on the configuration of the *Use Global Exception Handlers* (`CYGSEM_KERNEL_EXCEPTIONS_GLOBAL`) exception handling configuration suboption.

The `cyg_user_start` function, on line 20, is the main application entry point. The kernel startup process describing the `cyg_user_start` function is detailed in Chapter 5. To use the kernel API, the application must include the file `kapi.h`. The parameters passed into `system_call_exception_handler`, lines 9, 10, and 11, are:

- **data**—the parameter setup in the `cyg_exception_set_handler` call (0 in this example). It is often useful to use this parameter to pass a pointer to a structure for the exception handler routine to have information needed.
- **number**—the exception number that occurred.
- **info**—the processor-specific saved machine state. Currently, a pointer to the `HAL_SavedRegisters` structure is passed in this parameter. If this structure is modified, the saved state of the processor is altered. This allows the exception handler to correct the condition in order to allow the processor to continue. See the HAL macro definitions in Chapter 2 for an explanation of this architecture-specific structure.

If we look at Figure 3.1, the gray boxes illustrate the execution flow after setting up the exception handler in Code Listing 3.1.

The final routine that is called from the HAL is `restore_state`, as shown in Figure 3.1. This routine restores the processor registers to the state they were in prior to entering the default exception VSR. The state of the processor is contained in the `HAL_SavedRegisters` structure, which is passed in as a pointer. After `restore_state` returns, the processor can continue with its normal execution of the program.

3.1.2 Application Exception Handling

eCos provides a means for an application to completely take over all or some of the exception handling. This eliminates the HAL and kernel processing and allows the processor to vector directly to an application's VSR when an exception occurs. The application exception handler is then responsible for saving and restoring the processor's state, the same functionality provided by the HAL default exception VSR. It is also important to note that the VSR must be written in assembly language.

As we see in Figure 3.1, the white boxes illustrate how the application can take over exception handling. After the alignment exception occurs the VSR installed by the application is called to handle the exception condition. After the VSR completes, the handler must restore the processor's state so that the program can continue running.

The HAL defines macros to give the application access to the VSR table directly. These macros are described in Item List 3.2.

Item List 3.2 HAL Exception Vector Service Routine Macros

Syntax: **HAL_VSR_GET** (
 vector ,

```

        _pvsr_
    )

```

Parameters: `_vector_`—exception vector to retrieve from the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.

Description: `_pvsr_`—returned address of the VSR from the table.
Get the current VSR set in the `hal_vsr_table` and return it in the location pointed to by `_pvsr_`.

Syntax: **HAL_VSR_SET**(
 `_vector_`,
 `_vsr_`,
 `_poldvsr_`
)

Parameters: `_vector_`—exception vector to set in the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.
`_vsr_`—new vector service routine address to set in the VSR table.
`_poldvsr_`—returned address of the previously installed VSR.

Description: Replace the routine in the `hal_vsr_table` with the new vector service routine.

When installing a vector service routine, it is not necessary to call the HAL set and get macros directly. The eCos kernel API defines functions for the application to use instead. Item List 3.3 shows the kernel API functions for accessing the VSR table. These functions call the respective get and set HAL macros defined in Item List 3.3. For example, `cyg_interrupt_get_vsr` calls the HAL macro `HAL_VSR_GET`.

Item List 3.3 Kernel Exception Vector Service Routine API Functions

Syntax: `void`
cyg_interrupt_get_vsr(
 `cyg_vector_t` vector,
 `cyg_VSR_t` **vsr
);

Context: Any

Parameters: `vector`—exception vector to retrieve from the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.
`vsr`—returned pointer to vector service routine currently set in the VSR table.

Description: Return the pointer for the given exception vector from the VSR table.

Syntax: `void`
cyg_interrupt_set_vsr(
 `cyg_vector_t` vector,
 `cyg_VSR_t` *vsr
);

Context: Any

Parameters: `vector`—exception vector to set in the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.
`vsr`—address of the vector service routine to be set in the VSR table.

Description: Set the vector service routine directly into the VSR table. This vector service routine must be written in assembly and handle all exception processing, such as saving and restoring the processor state information.

3.2 Interrupts

An *interrupt* is an asynchronous external event that occurs during program execution, causing a break in the normal program execution. Typically, these external events are hardware related, such as a button press or timer expiration. Interrupts can happen at any time. Interrupts allow time-critical operations to be performed with higher precedence over normal program execution.

Similar to exception processing, when an interrupt occurs, the processor jumps to a specific address for execution of the *Interrupt Service Routine (ISR)*. Hardware support for interrupts varies among different architectures. Each processor has its own number of interrupt pins to trigger the ISR.

The methods for vector handling among different architectures are also diverse. Some processor architectures support vectored interrupts to individual vectors, while others have a single vector for all interrupts. When individual vectors are supported, an ISR can be attached directly to the vector for processing when the interrupt occurs. For single vector support, the software must determine which interrupt occurred before proceeding to the appropriate ISR.

One of the key concerns in embedded systems with respect to interrupts is *latency*. Latency is the interval of time from when an interrupt occurs until the ISR begins to execute. eCos employs an interrupt handling scheme to reduce interrupt latency in the system.

3.2.1 eCos Interrupt Model

eCos uses a split interrupt handling scheme where the interrupt processing is divided into two parts. The first part is the ISR, and the second part is the *Deferred Service Routine (DSR)*. This scheme allows for the minimal amount of interrupt latency in the system by reducing the amount of time spent inside interrupt service routines. The idea is to keep the processing done in the ISR to a bare minimum. In this scheme, the DSR is executed with interrupts enabled, allowing other higher priority interrupts to occur and be processed in the midst of servicing a lower priority interrupt.

In some cases, where very little interrupt processing needs to be done, the interrupt can be handled in the ISR completely. If more complex servicing is required, a DSR should be used. The DSR is executed at a later time when thread scheduling is allowed. Executing the DSR at a later time allows the DSR to use kernel synchronization mechanisms; for example, to signal a thread, via a semaphore, that an interrupt has occurred. However, there are periods where thread scheduling is disabled by the kernel, although these periods are kept as small as possible. User threads can suspend scheduling as well. This prevents the DSR from executing. Preventing the DSR from executing in a timely manner can lead to system failures from an interrupt source

overrunning. These issues should be kept in mind when designing the interrupt structure and how it interacts with the threads in the system.

In most cases, the DSR is executed immediately after the ISR completes. However, if a thread has locked the scheduler, the DSR is delayed until the thread unlocks it. The priority scheme is that ISRs have absolute priority over DSRs, and DSRs have absolute priority over threads.

NOTE eCos does not offer any configuration options to change the interrupt priority scheme among the ISR, DSR, and thread.

For the split interrupt handling scheme to work, the ISR must ensure the interrupt that just occurred does not recur until the DSR has finished its processing. To accomplish this, the ISR masks the current interrupt and the DSR will unmask the current interrupt after it has been serviced.

Some HAL implementations offer an interrupt nesting scheme. In this scenario, higher priority interrupts can interrupt, and be processed before, lower priority interrupts. A configuration option is available to enable this nesting functionality, as shown in Item List 3.5.

3.2.1.1 Interrupt and Scheduler Synchronization

It is important to understand the interaction between the interrupt (ISR and DSR) and the scheduler. Certain guidelines must be followed during interrupt processing to ensure proper operation.

First, ISRs cannot make any scheduler-related synchronization function calls. These include kernel API functions for semaphores, mutexes, and condition variables. Kernel synchronization functions cause interaction with the scheduler, which is disabled during ISR execution. Making such calls causes undefined behavior in the system and can lead to system failures. The eCos interrupt scheme allows these synchronization calls to be made from the DSR.

Typically, the DSR will execute immediately after the ISR finishes. DSRs are only delayed if the thread that was interrupted had locked the scheduler for some reason. Since the DSR executes when thread scheduling is enabled, certain kernel synchronization calls are allowed from within the DSR. This enables a DSR to wake up a thread for additional processing after the interrupt has occurred.

However, DSRs must not make a synchronization call that blocks. Blocking occurs when code execution must wait for a resource to be released. For example, using the kernel API call `cyg_mutex_lock` stops the current thread from executing until the mutex is released, with a `cyg_mutex_unlock` call.

NOTE It is worth bringing additional attention to this point about calling functions within a DSR. Any function that blocks *cannot* be called from within the DSR. This includes C library functions, such as `printf`, that might use blocking calls internally.

An example of a DSR that uses one of the kernel synchronization primitives is shown in Code Listing 3.2.

3.2.2 Interrupt Configuration

The eCos interrupt configuration options affect the way interrupts are processed in the system. Configuration options for interrupts are available at the HAL and kernel level. The configuration options available are architecture dependent.

The HAL-level configuration options are located under the *HAL Interrupt Handling* (CYGPKG_HAL_COMMON_INTERRUPTS) within the HAL Common Configuration Components. Item List 3.4 lists the configuration options available for setting up interrupts at the HAL level.

Item List 3.4 HAL Interrupt Configuration Options

Option Name	Use Separate Stack For Interrupts
CDL Name	CYGIMP_HAL_COMMON_INTERRUPTS_USE_INTERRUPT_STACK
Description	Allows a separate stack, maintained by the HAL, during interrupt processing. This eliminates the need for every thread stack to allow space for interrupt handlers. This option is enabled by default.
Option Name	Interrupt Stack Size
CDL Name	CYGNUM_HAL_COMMON_INTERRUPTS_STACK_SIZE
Description	Specifies, in bytes, the stack size for the interrupt stack. This is the stack that will be used for all interrupts when Use Separate Stack For Interrupts is enabled. The value for this option is dependent on the architecture and application, as well as other interrupt configuration options, such as interrupt nesting. This stack is also used during the HAL startup process. The default value for this option is 4096.
Option Name	Allow Nested Interrupts
CDL Name	CYGSEM_HAL_COMMON_INTERRUPTS_ALLOW_NESTING
Description	Causes the HAL default interrupt VSR to re-enable interrupts prior to invoking the interrupt handler. This allows other interrupts, typically of higher priority, to occur and be processed. This option is disabled by default.
Option Name	Save Minimum Context On Interrupt
CDL Name	CYGDBG_HAL_COMMON_INTERRUPTS_SAVE_MINIMUM_CONTEXT
Description	Permits the HAL interrupt handling code to use architecture-specific calling conventions, reducing the amount of state information saved. This improves performance and reduces code size. The drawback is that debugging is often more difficult. This option is enabled by default.
Option Name	Chain All Interrupts Together
CDL Name	CYGIMP_HAL_COMMON_INTERRUPTS_CHAIN
Description	Allows all interrupt vectors to be chained, requiring each handler to check if it needs to process the interrupt. The default for this option is disabled, allowing interrupts to be attached to individual vectors.
Option Name	Ignore Spurious Interrupts
CDL Name	CYGIMP_HAL_COMMON_INTERRUPTS_IGNORE_SPURIOUS
Description	Specifies whether to ignore an interrupt that might occur from the hardware source not being properly de-bounced or interrupts coming from glitches. This option is disabled by default.

Careful consideration is necessary when specifying the HAL interrupt configuration settings. Since the DSR executes with interrupts enabled, another interrupt might occur during the execution of the DSR. This means that the processor state information, which can be sizeable for certain architectures, for the current interrupt needs to be saved on the stack.

The stack used depends on the *Use Separate Stack For Interrupts* option. If this option is disabled, every thread in the system needs to be able to store the interrupt state information on its stack, since interrupts can happen at any time. If multiple interrupts occur, the stack would need the resources to hold the state information for each of these interrupts.

If *Use Separate Stack For Interrupts* is enabled, the threads in the system would only need to store a single interrupt state from the interrupt that caused the thread to de-schedule, along with its own thread state information. Only the interrupt stack has the responsibility to store this state information. The size of the interrupt stack should be carefully selected.

Another option that can change the way interrupts are handled is the configuration of *Allow Nested Interrupts*. The default setting for this option is disabled, meaning that interrupts are disabled during the ISR. When this option is enabled, the ISR enables interrupts to the processing of any higher priority interrupts that might occur. Some architectures support this feature in hardware. In most systems, the need to allow nested interrupts is nonexistent because the ISR is kept very short and the main work is accomplished in the DSR.

The kernel-level interrupt configuration options are under the *Kernel Interrupt Handling* (CYGPKG_KERNEL_INTERRUPTS) component. Within this component is the main option *Use Delayed Service Routines* (CYGIMP_KERNEL_INTERRUPTS_DSRS), which enables or disables the eCos split interrupt handling scheme.

NOTE If the *Use Delayed Service Routines* configuration option is disabled, it is up to the user to accommodate synchronization between ISRs and threads. Many of the kernel API functions cannot be called from an ISR context, as shown in the kernel API function tables throughout this book.

The kernel-level interrupt suboptions available when using DSRs are listed in Item List 3.5.

Item List 3.5 Kernel Interrupt Configuration Options

Option Name	Use Linked Lists For DSRs
CDL Name	CYGIMP_KERNEL_INTERRUPTS_DSRS_LIST
Description	Allows the kernel to keep track of pending DSRs in a linked list, preventing a table overflow from occurring. However, interrupts are disabled for a brief period while the kernel traverses the list. Alternatively, a fixed-size table can be used, with the number of entries configurable. This option is enabled by default.
Option Name	Use Fixed-Size Table For DSRs
CDL Name	CYGIMP_KERNEL_INTERRUPTS_DSRS_TABLE

Description	Enables the use of a fixed-size table for keeping track of pending DSRs in the kernel. If this option is enabled, the number of entries in the list is configurable as a suboption. This option is disabled by default.
Option Name	Chain All Interrupts Together
CDL Name	CYGIMP_KERNEL_INTERRUPTS_CHAIN
Description	Allows all interrupt vectors to be chained, requiring each handler to check if it needs to process the interrupt. The default for this option is disabled, allowing interrupts to be attached to individual vectors.

3.2.3 Interrupt Handling

An interrupt is a type of exception that can be found in a processor's exception table. As mentioned previously, different architectures support interrupts in various methods; however, eCos provides a standard method for handling interrupts across all HAL architectures. The ARM architecture is the exception that deviates from the standard eCos interrupt handling model.

The eCos interrupt handling mechanism, described in this section, can be bypassed by installing a vector service routine in the VSR table directly from the application layer. The application is then solely responsible for implementing, in assembly language, all support that would otherwise have been provided by the HAL default interrupt VSR, such as storing and restoring the processor's current state after the interrupt has been serviced.

We are going to take a look at the default configuration when the HAL default interrupt VSR is used. The HAL default interrupt VSRs for the different architectures are shown in Table 3.2. Figure 3.2 illustrates the execution flow of the eCos interrupt handling method. This figure shows the level—application, kernel, and HAL—responsible for executing each piece of code.

Table 3.2 Default Interrupt Vector Service Routines

Architecture	Default Interrupt Vector Service Routine Name
CalmRISC16 ^a	__default_irq_vsr and __default_fiq_vsr
Fujitsu FR-V	_interrupt
i386 (including Synth), PowerPC, SuperH	cyg_hal_default_interrupt_vsr
Hitachi H8/300, MIPS, MN10300	__default_interrupt_vsr
V8x	do_interrupt
SPARC, SPARClike	hal_default_interrupt_vsr

^a The CalmRISC16 processor contains two different default exception routines, one for a swi exception and one for a trace exception.

The following is a step-by-step description of the execution flow shown in Figure 3.2; the numbers correspond to the events in the diagram.

1. The first item in Figure 3.2 shows the execution of a thread.
2. The next event is an external hardware interrupt.
3. Now, the processor looks into the VSR table, `hal_vsr_table`, to determine the location of the interrupt vector service routine to execute. During HAL startup, `hal_mon_init` installs the default interrupt VSR into the VSR table for the external interrupt. The names of the default interrupt VSRs for the different architectures are shown in Table 3.2. The ARM architecture is not shown in the table because it defines separate vector service routines for each of its interrupts.
4. Next, the default interrupt VSR begins executing. The first task of the default interrupt VSR is to save the current processor state. As mentioned before, the current processor's state can be saved either on a thread's stack or on the separate interrupt stack, depending on the HAL interrupt configuration options selected.

After the processor's state information has been stored, the default interrupt VSR increments the `cyg_scheduler_sched_lock` kernel variable to ensure that scheduling does not take place.

Next, the default vector service routine needs to find out what ISR to call. ISRs are installed by the application, as we see in the example in Code Listing 3.2.

The HAL uses three tables, implemented as arrays, to maintain the ISR information needed. The size of these tables is architecture specific. The ISR tables are:

- `hal_interrupt_handlers`—contains the addresses of the interrupt service routines installed by the application.
- `hal_interrupt_data`—contains the data to be passed into the ISR.
- `hal_interrupt_objects`—contains information that is used at the kernel level and hidden from the application layer.

The HAL default interrupt VSR uses an architecture-specific function, `hal_intc_decode`, to perform the lookup into the `hal_interrupt_handlers` table. This function finds the index into the table based on the interrupt vector number and/or through examining the hardware, such as an interrupt controller. The value is used for indexing into the data (which is passed into the ISR) and objects (used at the kernel level) tables as well.

5. Next, the default VSR calls the ISR installed by the application. The ISR, which executes at the application level, performs any necessary functions for the particular interrupt. The ISR notifies the kernel that the DSR should be posted for execution by returning `CYG_ISR_CALL_DSR`. The ISR also returns `CYG_ISR_HANDLED` to terminate any chained interrupt processing. An example of an application ISR and DSR is shown in Code Listing 3.2.

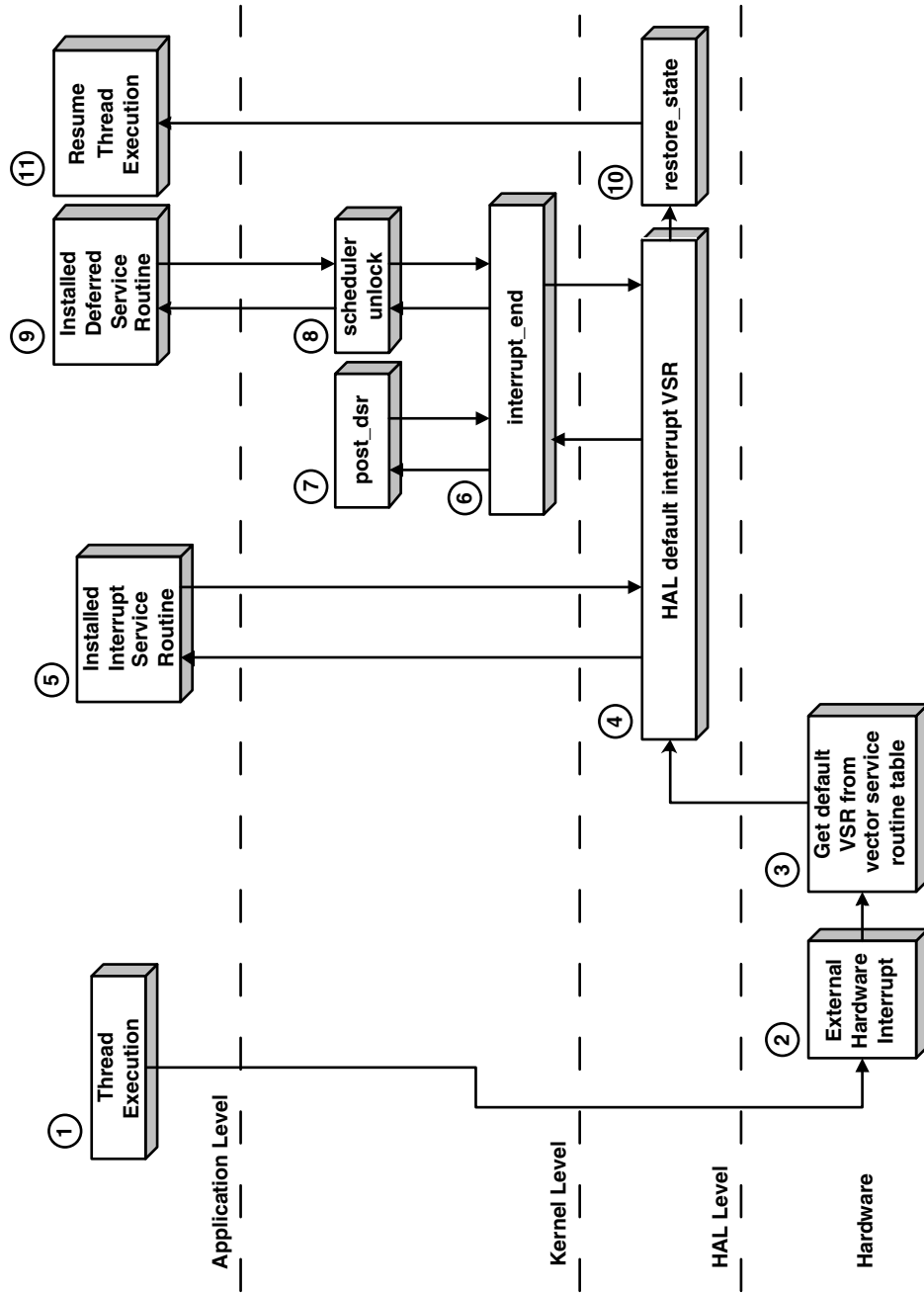


Figure 3.2 eCos interrupt handling execution flow.

6. After the ISR returns, the default interrupt VSR calls the function `interrupt_end`. The function `interrupt_end` is part of the kernel interrupt handling code and can be found in the file `intr.cxx` under the `kernel\current\src\intr` subdirectory.
7. If a DSR needs to execute, `interrupt_end` calls the function `post_dsr`. `post_dsr`, also part of the kernel interrupt handling code in the file `intr.cxx`, manages the implementation of posting the DSR. The two kernel interrupt configuration options *Use Linked Lists For DSRs* and *Use Fixed-Size Table For DSRs* determine the structure for the DSR list. See the *Interrupt Configuration* section in this chapter for a detailed description of these options.
8. After `post_dsr` returns, `interrupt_end` then unlocks the scheduler.
9. If the scheduler lock variable is at 0, the DSR executes. If the scheduler lock variable is greater than 0, then the thread that locked the scheduler executes at this point. In our case, we assume another thread has not locked the scheduler. An example of a DSR is shown in Code Listing 3.2.
After the DSR completes, the scheduler unlock code continues execution.
10. Next, the HAL default interrupt VSR restores the processor state. This takes place in the routine `restore_state`, the same function called from the default exception VSR.
11. Finally, thread execution continues.

NOTE It is important to realize that the thread interrupted, as described in step 1, might not be the thread that is restored for execution by the scheduler, detailed in step 11. This can happen if the DSR executes, as described in step 8, and causes another thread to become ready to run that is at a higher priority than the thread that was interrupted.

For example, let's say that two threads are created in a system—thread A and thread B—where thread B is at a higher priority than thread A. Thread B waits on a semaphore indicating that data has arrived from a device. Now suppose that thread A is currently executing when the device causes an interrupt, indicating data has arrived, as shown in step 2. Steps 3 through 8 occur as described. However, in step 9, the DSR associated with this interrupt posts the semaphore that thread B is waiting on. When the scheduler unlock code continues execution, the context of thread B is restored because it is at a higher priority than thread A. Therefore, in step 11, thread B executes.

Let's now look at ISR and DSR functions. Both the ISR and DSR are set up in the application using the kernel API function calls. Code Listing 3.2 shows an example of ISR and DSR functions and how the kernel API is used to set them up.

```
1 #include <cyg/kernel/kapi.h>
2
```

```
3 static cyg_interrupt int1;
4 static cyg_handle_t int1_handle;
5 static cyg_sem_t data_ready;
6
7 #define CYGNUM_HAL_INTERRUPT_1 1
8 #define CYGNUM_HAL_PRI_HIGH 0
9
10 //
11 // Interrupt service routine for interrupt 1.
12 //
13 cyg_uint32 interrupt_1_isr(
14     cyg_vector_t vector,
15     cyg_addrword_t data)
16 {
17     // Block this interrupt from occurring until
18     // the DSR completes.
19     cyg_interrupt_mask( vector );
20
21     // Tell the processor that we have received
22     // the interrupt.
23     cyg_interrupt_acknowledge( vector );
24
25     // Tell the kernel that chained interrupt processing
26     // is done and the DSR needs to be executed next.
27     return( CYG_ISR_HANDLED | CYG_ISR_CALL_DSR );
28 }
29
30 //
31 // Deferred service routine for interrupt 1.
32 //
33 void interrupt_1_dsr(
34     cyg_vector_t vector,
35     cyg_ucount32 count,
36     cyg_addrword_t data)
37 {
38     // Signal the thread to run for further processing.
39     cyg_semaphore_post( &data_ready );
40
41     // Allow this interrupt to occur again.
42     cyg_interrupt_unmask( vector );
43 }
44
45 //
46 // Main starting point for the application.
47 //
48 void cyg_user_start(
49     void)
50 {
```

```
51     cyg_vector_t int1_vector = CYGNUM_HAL_INTERRUPT_1;
52     cyg_priority_t int1_priority = CYGNUM_HAL_PRI_HIGH;
53
54     // Initialize the semaphore used for interrupt 1.
55     cyg_semaphore_init( &data_ready, 0 );
56
57     //
58     // Create interrupt 1.
59     //
60     cyg_interrupt_create(
61         int1_vector,
62         int1_priority,
63         0,
64         &interrupt_1_isr,
65         &interrupt_1_dsr,
66         &int1_handle,
67         &int1);
68
69     // Attach the interrupt created to the vector.
70     cyg_interrupt_attach( int1_handle );
71
72     // Unmask the interrupt we just configured.
73     cyg_interrupt_unmask( int1_vector );
74 }
```

Code Listing 3.2 Example using the eCos kernel APIs for installing an interrupt within an application.

As we see in Code Listing 3.2, the main function, `cyg_user_start` on line 48, is called during the kernel startup procedure. We go through the kernel startup procedure in Chapter 5. After initializing the semaphore, see line 55, which is used in the DSR, the interrupt is created for the interrupt vector `CYGNUM_HAL_INTERRUPT_1`, using the kernel function `cyg_interrupt_create`, on line 60. The interrupt vector (`CYGNUM_HAL_INTERRUPT_1` on line 51) and priority (`CYGNUM_HAL_PRI_HIGH` on line 52) are defined on lines 7 and 8, respectively. The data field, as we see on line 63, is set to zero since we do not need to pass anything into the ISR or DSR.

Next, the main function handles attaching the interrupt just created to the vector using `cyg_interrupt_attach`, shown on line 70. Finally, the interrupt 1 vector is unmasked, by `cyg_interrupt_unmask` on line 73, so that it can be processed when global interrupts are enabled. Global interrupts are enabled manually or, as in this example, when the scheduler starts after `cyg_user_start` returns.

The ISR, `interrupt_1_isr` on line 13, masks the current interrupt vector, shown on line 16, to ensure that it does not occur before the DSR has finished. It then acknowledges the interrupt within the processor, as we see on line 23. Lastly, the ISR signals to the kernel to post

the DSR for further processing by returning `CYG_ISR_CALL_DSR`, on line 27. The ISR also returns `CYG_ISR_HANDLED` to terminate any chained interrupt processing.

When the DSR executes, `interrupt_1_dsr` on line 33, it signals a semaphore, as we see on line 39, and then unmask the current interrupt, shown on line 42, so it can occur again. The semaphore, `data_ready`, used in this example demonstrates a method of synchronization between an interrupt and a thread. The thread, which is not included in the code listing, waits on this semaphore before beginning its processing. You can find the kernel API function descriptions for semaphores, including an example using the semaphore kernel API, in Chapter 6, *Threads and Synchronization Mechanisms*.

Detailed descriptions of the kernel API functions used in Code Listing 3.2, and the underlying HAL macros, are found in this chapter in the *Interrupt Control* section.

NOTE In some circumstances it might be more useful to explicitly post a DSR, or multiple DSRs, instead of using the `CYG_ISR_CALL_DSR` return value from the ISR. An ISR can explicitly post a DSR by calling `cyg_interrupt_post_dsr` and passing in the interrupt object returned from the `cyg_interrupt_create` function call, `intr`. This allows a single ISR to trigger multiple DSRs, or for an ISR to determine, at the time of the interrupt, which specific DSR to run from a list of possible DSRs.

The function `cyg_interrupt_post_dsr` is not part of the published kernel interrupt API. The function is implemented in the file `intr.cxx` in the `intr` subdirectory under the `kernel` source code directory. To use this function, it must be declared as follows:

```
extern void
cyg_interrupt_post_dsr(CYG_ADDRWORD
intr_handle);
```

Attaching different DSRs to a single interrupt is accomplished by creating multiple interrupt objects that have the same ISR with a different DSR. In the call to the function `cyg_interrupt_create`, different interrupt objects are passed in the `intr` parameter along with different DSRs in the `dsr` parameter. However, the same ISR is passed in the `isr` parameter. It is important to understand that the function `cyg_interrupt_attach` only needs to be called once, with one of the interrupt handles created for the ISR. See Item List 3.7 for details about the kernel API for interrupts.

3.2.4 Interrupt Control

The HAL and kernel offer control over interrupt configuration. The HAL controls interrupts through the use of macros. The macro names are common across all HAL architectures; however, the implementations of these macros are architecture specific. The HAL macros are defined in the file `hal_intr.h` under the `arch` subdirectory.

The kernel API contains interrupt control functions that make use of these macros. Therefore, it is not necessary for an application to call directly into the HAL; rather, it can use the kernel API to configure system interrupts. The HAL and kernel interrupt control functionality is broken down into three groups:

- Interrupt Service Routine Management
- Interrupt State Management
- Interrupt Controller Management

NOTE It is important to use the kernel API functions and avoid using the HAL macros directly. Using the kernel API guarantees consistency over the underlying HAL macro implementations. An example of maintaining consistency is when using the configuration option to chain interrupts together. By going through the kernel API when attaching interrupts (`cyg_interrupt_attach`), rather than directly using the `HAL_INTERRUPT_ATTACH` macro, the algorithm for inserting the interrupt in the chain list is handled by the kernel API function. This ensures that the interrupts are inserted into the list using the process each time.

3.2.4.1 Interrupt Service Routine Management

The first of the three groups, *Interrupt Service Routine Management*, controls the attachment and detachment of interrupt service routines within the three HAL ISR tables (handlers, data, and objects). The HAL ISR Management macros are described in Item List 3.6.

Item List 3.6 HAL Interrupt Service Routine Management Macros

Syntax: **HAL_INTERRUPT_ATTACH** (
 vector,
 isr,
 data,
 object
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts. This value is used to determine the index into the ISR tables.

`_isr_`—ISR address to install into the handler table.

`_data_`—data address to install into the data table.

`_object_`—object address to install into the object table.

Description: Places the parameters into the ISR tables for the given interrupt vector. The ISR is called from the HAL default VSR when the interrupt occurs, the data and object are passed into the ISR as parameters.

Syntax: **HAL_INTERRUPT_DETACH**(
 vector,
 isr
);

Parameters: *_vector_*—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts. This value is used to determine the index into the ISR tables.
isr—ISR address to remove from the handler table.

Description: Remove the interrupt service routine from the ISR table. The ISR is then set to a HAL default ISR. The data and object tables are set to zero for this vector index.

The kernel API functions for Interrupt Service Routine Management are listed in Item List 3.7.

Item List 3.7 Kernel Interrupt Service Routine Management API Functions

Syntax: void
cyg_interrupt_create(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t *isr,
 cyg_DSR_t *dsr,
 cyg_handle_t *handle,
 cyg_interrupt *intr
);

Context: Thread

Parameters: *vector*—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.
priority—for certain HAL architectures, an interrupt priority level is supported. This value is used when the interrupt is attached.
data—data address to install into the data table when the interrupt is attached. Often, this parameter is used to provide context information when an ISR is attached to multiple interrupt sources. For example, the parameter can be a pointer to a structure containing state information for the interrupt source identifying which interrupt triggered.
isr—address of the interrupt service routine to install when the interrupt is attached.
dsr—address of the deferred service routine to install.
handle—pointer to the new interrupt.
intr—returned address of the new interrupt object.

Description: Construct an interrupt object in memory. The interrupt is not attached to the vector, however, until `cyg_interrupt_attach` is called.

Syntax: void
cyg_interrupt_delete(
 cyg_handle_t interrupt
);

Context: Thread

- Parameters: `interrupt`—handle to the interrupt.
- Description: Remove the interrupt object from memory, freeing the memory passed in the `intr` parameter to `cyg_interrupt_create`. This call also detaches, using the `HAL_INTERRUPT_DETACH` macro, the ISR, data, and object from the ISR tables.
- Syntax:

```
void
cyg_interrupt_attach(
    cyg_handle_t interrupt
);
```
- Context: Thread
- Parameters: `interrupt`—handle to the interrupt.
- Description: Attach the interrupt to the vector allowing interrupts to be delivered to the ISR. This function makes use of the `HAL_INTERRUPT_SET_LEVEL` and `HAL_INTERRUPT_ATTACH` macros. The interrupt is also set up in the chain list if the configuration option is enabled.
- Syntax:

```
void
cyg_interrupt_detach(
    cyg_handle_t interrupt
);
```
- Context: Thread
- Parameters: `interrupt`—handle to the interrupt.
- Description: Detach the interrupt from the vector preventing interrupts from being delivered to the ISR. This function calls the `HAL_INTERRUPT_DETACH` macro and removes the interrupt from the chain list if the configuration option is enabled.

3.2.4.2 Interrupt State Management

The second group, *Interrupt State Management*, allows control over the state of the processor's interrupt mask mechanism by accessing the global interrupt enable found in the processor's register. These functions do not access the interrupt controller, which might be present on certain variants. The HAL Interrupt State Management macros are described in Item List 3.8.

Item List 3.8 HAL Interrupt State Management Macros

- Syntax:

```
HAL_DISABLE_INTERRUPTS(
    _old_
);
```
- Parameters: `_old_`—returned state of the interrupt mask.
- Description: Disable all interrupts. This is accomplished by using the interrupt enable found in one of the processor's registers.
- Syntax:

```
HAL_ENABLE_INTERRUPTS( )
```
- Description: Enable all interrupts. This is accomplished by using the interrupt enable found in one of the processor's registers.

Syntax: **HAL_RESTORE_INTERRUPTS** (
 old
)
Parameters: _old_—state of the interrupt mask.
Description: Restore the interrupts according to the state of the interrupt mask specified.

Syntax: **HAL_QUERY_INTERRUPTS** (
 old
)
Parameters: _old_—returned state of the interrupt mask.
Description: Determine state of interrupt mask and return the value.

The kernel API functions for Interrupt State Management are listed in Item List 3.9.

Item List 3.9 Kernel Interrupt State Management API Functions

Syntax: void
 cyg_interrupt_disable (
 void
);
Context: Any
Parameters: None
Description: Disable all interrupts, using the HAL_INTERRUPT_DISABLE macro.

Syntax: void
 cyg_interrupt_enable (
 void
);
Context: Any
Parameters: None
Description: Enable all interrupts, using the HAL_INTERRUPT_ENABLE macro.

3.2.4.3 Interrupt Controller Management

The final group, *Interrupt Controller Management*, provides control over any interrupt controller that might be present for a specific variant. Not all HAL architectures have an interrupt controller.

A specific platform or variant that does contain an interrupt controller provides the implementation of these macros in its own interrupt definition file. The names of the platform or variant files containing these override macros differ among architectures. The HAL Interrupt Controller Management macros are defined in Item List 3.10.

Item List 3.10 HAL Interrupt Controller Management Macros

Syntax: **HAL_INTERRUPT_MASK** (
 vector
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Block the designated interrupt from occurring. This is typically a platform or variant specific implementation, which requires masking interrupts in the processor's registers.

Syntax: **HAL_INTERRUPT_UNMASK** (
 `_vector_`
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Unblock the designated interrupt. This is typically a platform- or variant-specific implementation, which requires masking interrupts in the processor's registers.

Syntax: **HAL_INTERRUPT_ACKNOWLEDGE** (
 `_vector_`
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Acknowledges the current interrupt from the specified vector. This informs the processor that the interrupt was received and resets the interrupt to an inactive state. This is typically a platform- or variant-specific implementation, where the HAL modifies the processor's interrupt acknowledge bit in a register.

Syntax: **HAL_INTERRUPT_CONFIGURE** (
 `_vector_`,
 `_level_`,
 `_up_`
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

`_level_`—specifies whether the interrupt is level- or edge-triggered.

`_up_`—specifies a falling or rising edge for edge-triggered interrupts and a high or low level for level-triggered interrupts.

Description: Programs the interrupt controller with the configuration settings for a specified interrupt. These settings determine the method for detecting an interrupt. Not all HAL architectures support these configuration settings.

Syntax: **HAL_INTERRUPT_SET_LEVEL** (
 `_vector_`,
 `_level_`
)

Parameters: `_vector_`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.
`_level_`—priority level setting for the specified interrupt.

Description: Configures the priority level of the specified interrupt. Not all HAL architectures support these configuration settings.

The kernel API functions for the Interrupt Controller Management group are defined in Item List 3.11.

Item List 3.11 Kernel Interrupt Controller Management API Functions

Syntax: `void`
`cyg_interrupt_mask(
 cyg_vector_t vector
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Program the interrupt controller to block delivery of interrupts for the vector specified. This function calls the `HAL_INTERRUPT_MASK` macro. All interrupts are disabled during this function call.

Syntax: `void`
`cyg_interrupt_mask_intunsafe(
 cyg_vector_t vector
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Program the interrupt controller to block delivery of interrupts for the vector specified. This function can be called when interrupts are currently disabled.

Syntax: `void`
`cyg_interrupt_unmask(
 cyg_vector_t vector
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Program the interrupt controller to unblock interrupts for the specified vector, allowing interrupts to be delivered to the ISR. This function calls the `HAL_INTERRUPT_UNMASK` macro. All interrupts are disabled during this function call.

Syntax: `void
cyg_interrupt_unmask_intunsafe(
 cyg_vector_t vector
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Program the interrupt controller to unblock interrupts for the specified vector, allowing interrupts to be delivered to the ISR. This function can be called when interrupts are currently disabled.

Syntax: `void
cyg_interrupt_acknowledge(
 cyg_vector_t vector
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.

Description: Acknowledge the current interrupt from the specified vector. This function is called from within the ISR to cancel the interrupt request from the processor, preventing a re-trigger of the same interrupt. This function calls the `HAL_INTERRUPT_ACKNOWLEDGE` macro.

Syntax: `void
cyg_interrupt_configure(
 cyg_vector_t vector,
 cyg_bool_t level,
 cyg_bool_t up
);`

Context: Any

Parameters: `vector`—HAL architecture-specific interrupt definition number. Each HAL defines, in the file `hal_intr.h`, all interrupts supported starting from 0. The platform or variant can define additional interrupts.
`level`—specifies whether the interrupt is level or edge triggered.
`up`—specifies a falling or rising edge for edge-triggered interrupts, and a high or low level for level-triggered interrupts.

Description: Program the interrupt controller with the method for detecting an interrupt. This function calls the `HAL_INTERRUPT_CONFIGURE` macro.

The kernel API functions for Symmetric Multi-Processing (SMP) systems are defined in Item List 3.12. eCos support for SMP is covered in Chapter 8.

Item List 3.12 Kernel SMP Interrupt API Functions

Syntax: `void
cyg_interrupt_set_cpu(
 cyg_vector_t vector,`

```
        cyg_cpu_t cpu
    );
```

Context: Any

Parameters: `vector`—exception vector to retrieve from the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.

Description: `cpu`—processor to receive interrupt. Specifies the processor to handle the hardware interrupt. The VSR, ISR, and possibly DSR execute on this processor as well.

Syntax: `cyg_cpu_t`
`cyg_interrupt_get_cpu`(
 `cyg_vector_t vector`
);

Context: Any

Parameters: `vector`—exception vector to set in the VSR table. Each HAL defines, in the file `hal_intr.h`, all exceptions supported starting from 0. This value is used to determine the index into the VSR table.

Description: Returns the CPU designated to handle the specified interrupt.

3.3 Summary

In this chapter, we looked at eCos exception handling. We saw how the VSR table is used by eCos to run exception handling routines. We looked at the two options for exception handling. One option allows the HAL and kernel to provide default exception routines. The other option requires complete support of exception handling to be provided by the application.

Then we proceeded to examine the eCos split interrupt processing scheme using ISRs and DSRs. This provides us with a mechanism to reduce interrupt latency and allows interrupts to synchronize with the threads running in the system. Finally, we went through the eCos interrupt handling process.

Virtual Vectors

This chapter details the mechanism used by the eCos HAL for various communication methods called virtual vectors. We cover how virtual vectors are used, the services provided, and the different configuration options available. Understanding virtual vectors aids our understanding for the different debugging methods described in later chapters.

4.1 Virtual Vectors

eCos defines a group of pointers to service functions and data called *virtual vectors*. The principal role of the virtual vectors is to allow services provided by a ROM startup configuration, such as a ROM monitor, to be accessed by a RAM startup configuration, the application being debugged.

For example, during typical application development, the target hardware boots up using a ROM monitor, such as RedBoot. The application being debugged is built using a RAM startup type. Then, the host running a debugger, such as GDB, uses a communication channel to download the application software and exchange debug information with RedBoot. It is also useful for the application to use this same communication channel for diagnostic messages, such as `diag_printf`. The application's diagnostic code needs to be aware of the communication channel to use in order to output information. Therefore, either RedBoot needs to pass the communication channel information to the application during startup, or a level of indirection can be used. The level of indirection is a virtual vector, which offers a more general solution. Using this type of configuration also eliminates the need for debug code from the user application because the ROM monitor provides this functionality. Additional information about the RedBoot ROM monitor can be found in Chapter 9, *The RedBoot ROM Monitor*.

During typical development, the application does not need to be aware of the virtual vectors; diagnostic output is seamlessly routed to the appropriate communication channel.

Using virtual vectors also makes it possible to debug user applications from an arbitrary channel. For example, if an application only contains a device driver for a serial port for its own communications, a ROM monitor can use an Ethernet port, with proper networking support, for debug communications. The user application does not need to contain an Ethernet device driver or networking stack because this is all handled by the ROM monitor. Since this Ethernet port debugging functionality can be eliminated from the user application, the result is a smaller image that only contains code used in the production release.

One issue with sharing resources between a ROM monitor and a user application is that the two are linked separately; therefore, each is unaware of the location of objects in the other's address space. Virtual vectors are used to overcome this problem by providing a common structure with a defined layout that is known by the ROM monitor and user application.

Virtual vectors are contained within the *Virtual Vector Table* (VVT). The VVT is then placed at a static memory location in the target address space, of which both the RAM application and ROM monitor are aware. The VVT, defined as `hal_virtual_vector_table` in the file `hal_if.h` under the `common HAL` subdirectory, is an array of 64 vectors. The actual location of the VVT is dependent on the HAL architecture and setup in the linker script file. Linker script files are located under the `arch` subdirectory and have a `.ld` extension. The memory, 256 bytes, for the VVT is allocated whether or not the VVT is used.

The method for using the VVT varies depending on the functionality needed by the RAM application. Functions in the VVT can be implemented in the ROM monitor, the RAM application, disabled by installing pointers to dummy routines at certain locations, or control can be taken over at run time by reinitializing certain pointers in the VVT.

In general, a loose policy for governing the VVT is that the ROM monitor or the standalone application initializes all vectors in the table. The RAM application can then reinitialize any services it needs to provide. The default configuration is that the ROM monitor provides the console and debugging I/O services, and the RAM application initializes all other services.

NOTE Some platforms do not use virtual vectors. However, it is recommended that all new ports implement virtual vector functionality.

Table 4.1 lists the available virtual vectors contained in the VVT. The virtual vector numbers, which are defined in the file `hal_if.h` under the `common HAL` subdirectory, correspond to the location of the vector in the VVT. Service functions that are unused are set to the `nop_service` function, which returns 0, defined in the `common HAL`. Data services that are unused are set to 0.

Table 4.1 Virtual Vector Table Service Functions and Data

Service Function or Data	Virtual Vector Number	Description
Virtual Vector Table Version	0	Version of the table. This value contains the total number of virtual vectors in the upper 16 bits, and the definition number of the last supported virtual vectors in the lower 16 bits. For this VVT, the total number of virtual vectors is 64d (0x40), and the definition number of the last virtual vector, Flash ROM Configuration, is 20d (0x14). The version is therefore 0x4014.
Interrupt Table	1	Interrupt service routine table, <code>hal_interrupt_handlers</code> , address.
Exception Table	2	Exception vector service routine table, <code>hal_vsr_table</code> , address.
Debug Vector	3	UNUSED
Kill Vector	4	Function to execute when a kill instruction is received from the debugger. This typically calls the platform-specific reset function.
Console I/O Procedures Table	5	Communication interface procedures table used for console I/O. This is described in further detail in the <i>Communication Channels</i> section of this chapter.
Debug I/O Procedures Table	6	Communication interface procedures table used for debugging I/O. This is described in further detail in the <i>Communication Channels</i> section of this chapter.
Flush Data Cache	7	Flush processor data cache for a specified region. Uses the HAL macros <code>HAL_DCACHE_FLUSH</code> and <code>HAL_DCACHE_INVALIDATE</code> .
Flush Instruction Cache	8	Flush processor instruction cache for a specified region. Uses the HAL macros <code>HAL_ICACHE_FLUSH</code> and <code>HAL_ICACHE_INVALIDATE</code> .
CPU Data	9	UNUSED

Table 4.1 Virtual Vector Table Service Functions and Data *(Continued)*

Service Function or Data	Virtual Vector Number	Description
Board Data	10	UNUSED
System Information	11	UNUSED
Set Debug Communication Channel	12	Sets the current debug communication channel.
Set Console Communication Channel	13	Sets the current console communication channel.
Set Serial Baud Rate	14	UNUSED
Debug System Call	15	Communication vector between ROM monitor and RAM application. The ROM monitor uses this function, which is provided by the RAM application, to retrieve debug data about the application, such as thread information.
Reset	16	Performs a software reset.
Console Interrupt Flag	17	This flag is set when a debugger interrupt is detected during the processing of console I/O.
Microsecond Delay	18	Delay by the specified number of microseconds.
Debug Data	19	UNUSED
Flash ROM Configuration	20	Allows an application to access the Flash ROM configuration data in the ROM monitor. The information contained in the configuration is monitor specific, but can include the Ethernet Media Access Control (MAC) address, for example.
Install Breakpoint	35	Installs a breakpoint at a specified address, which is used by asynchronous breakpoint support for GDB.

4.1.1 Virtual Vector Configuration

The virtual vector configuration options affect the initialization of the VVT. The default configuration options for the VVT, which can be overridden, are that the ROM monitor provides the debugging and diagnostic I/O services, and RAM applications rely on these services. In the case of a standalone production system, all services are provided by the application.

The virtual vector configuration options are located under the *ROM Monitor Support* (CYGPKG_HAL_ROM_MONITOR) configuration option within the HAL Common Configuration Components. The main virtual vector configuration option is *Enable Use of Virtual Vector Calling Interface* (CYGSEM_HAL_VIRTUAL_VECTOR_SUPPORT), which is enabled by default and can only be disabled by hand-editing configuration files. Item List 4.1 lists the virtual vector configuration suboptions.

The two configuration options, *Behave as a ROM Monitor* (CYGSEM_HAL_ROM_MONITOR) and *Work With a ROM Monitor* (CYGSEM_HAL_USE_ROM_MONITOR), determine the type of image being built. These options are also found under the *ROM Monitor Support* (CYGPKG_HAL_ROM_MONITOR) configuration option within the HAL Common Configuration Components. The two ROM monitor options dictate where the virtual vector configuration suboption settings take effect. For RAM application debugging, typically *Work With a ROM Monitor* is enabled. If a ROM monitor or released application is being built, *Behave as a ROM Monitor* is selected.

Item List 4.1 Virtual Vector Configuration Suboptions

Option Name	Inherit Console Settings From ROM Monitor
CDL Name	CYGSEM_HAL_VIRTUAL_VECTOR_INHERIT_CONSOLE
Description	Allows the RAM application to inherit the console setup by the ROM monitor using the configured channel and text encoding style.
Option Name	Debug Channel Is Configurable
CDL Name	CYGPRI_HAL_VIRTUAL_VECTOR_DEBUG_CHANNEL_CONFIGURABLE
Description	Allows the HAL startup code to make use of the debug channel configuration.
Option Name	Console Channel Is Configurable
CDL Name	CYGPRI_HAL_VIRTUAL_VECTOR_CONSOLE_CHANNEL_CONFIGURABLE
Description	Allows the HAL startup code to make use of the console channel configuration.
Option Name	Initialize Whole Virtual Vector Table
CDL Name	CYGSEM_HAL_VIRTUAL_VECTOR_INIT_WHOLE_TABLE
Description	Causes the entire VVT to be initialized a default service function, <code>nop_service</code> . This is performed in <code>hal_if_init</code> .
Option Name	Claim Virtual Vector Table Entries By Default
CDL Name	CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_DEFAULT

Description Allows the image to provide all services in the VVT, except Debug and Console Communication services, which will be provided by the ROM monitor. This option enables or disables the claiming of the individual virtual vector configuration options.

Option Name **Claim Reset Virtual Vectors**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_RESET

Description Allows the image to provide the Reset and Kill Vector services.

Option Name **Claim DELAY_US Virtual Vector**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_DELAY_US

Description Allows the image to provide the Microsecond Delay service.

Option Name **Claim Cache Virtual Vectors**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_CACHE

Description Allows the image to provide the Instruction and Data Cache Flush services.

Option Name **Claim Data Virtual Vectors**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_DATA

Description Allows the image to provide the Data services, which are currently unused in the VVT.

Option Name **Claim COMMS Virtual Vectors**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_COMMS

Description Allows the image to provide the Debug and Console Communication Channels.

Option Name **Do Diagnostic IO Via Virtual Vector Table**

CDL Name CYGSEM_HAL_VIRTUAL_VECTOR_DIAG

Description Allows all HAL-level I/O to be performed using the configuration settings in the VVT. This option is always enabled on platforms that contain virtual vector support.

4.1.2 Virtual Vector Table Initialization

The common HAL defines macros, in the file `hal_if.h`, to execute and set the services within the VVT, `hal_virtual_vector_table`. There are two macros defined for each virtual vector, a *call macro*—which executes the service in the VVT, and a *set macro*—which sets the service in the VVT. Code Listing 4.1 shows the call and set macros for the Reset virtual vector. The call macro has the form `CYG_ACC_CALL_IF_XXX`, as we see on lines 1 and 2, and set macro has the form `CYG_ACC_CALL_IF_XXX_SET`, shown on lines 4 and 5, where XXX is the defined name of the virtual vector that gives its location in the VVT. The number of parameters passed into the call macro varies depending on the virtual vector service function. The set macro always has a single value passed in, the address of the service function or the data value to set in the VVT.

```
1 #define CYGACC_CALL_IF_RESET(_p_, _n_) \
2 (hal_virtual_vector_table[CYGNUM_CALL_IF_RESET])(_p_), (_n_)
```

```
3
4 #define CYGACC_CALL_IF_RESET_SET(_x_) \
5 hal_virtual_vector_table[CYGNUM_CALL_IF_RESET] = (CYG_ADDRWORD) (_x_)
```

Code Listing 4.1 Common HAL VVT call and set macros for Reset virtual vector.

All HAL initialization sequences, whether running from a ROM monitor, a ROM application, or RAM application, call the function `hal_if_init`, located in the file `hal_if.c` under the `common` HAL subdirectory. Within this function, the initialization sequence for the VVT is determined by the virtual vector configuration suboptions selected and the type of image, ROM monitor/application or RAM application, the eCos library is built for use with.

NOTE It is not possible to step through the `hal_if_init` function if the RAM application is configured to initialize the whole VVT or reconfigure the communication channels. This scenario would create a problem with the RAM application HAL reinitializing services in the VVT while the ROM monitor is trying to use the services already configured in the VVT for debugging I/O. One way to debug this scenario is to leave the VVT that the ROM monitor is using alone and use a RAM application version of the VVT, at another address, for debugging the RAM application code.

In a typical eCos debug environment, two separate images exist, a ROM monitor and a RAM application. Using these two images and the default virtual vector configuration suboptions, the steps involved in the initialization of the VVT are:

1. ROM monitor is booted, causing the HAL built into the ROM monitor image to initialize the VVT with its own default service vectors according to the virtual vector configuration suboption settings.
2. The RAM application is loaded into memory via the debug channel in the ROM monitor.
3. Next, the ROM monitor is given a command to execute the RAM application. This turns control over to the RAM application; however, the VVT still contains function and data services provided by the ROM monitor.
4. Finally, the RAM application HAL executes, reinitializing the VVT according to the virtual vector configuration suboption settings. Any services that the RAM application provides are set into the VVT.

Figure 4.1 shows the initialization sequence and default VVT after the common HAL has performed its setup. The functions in the VVT, shown in Figure 4.1, are implemented in the common HAL.

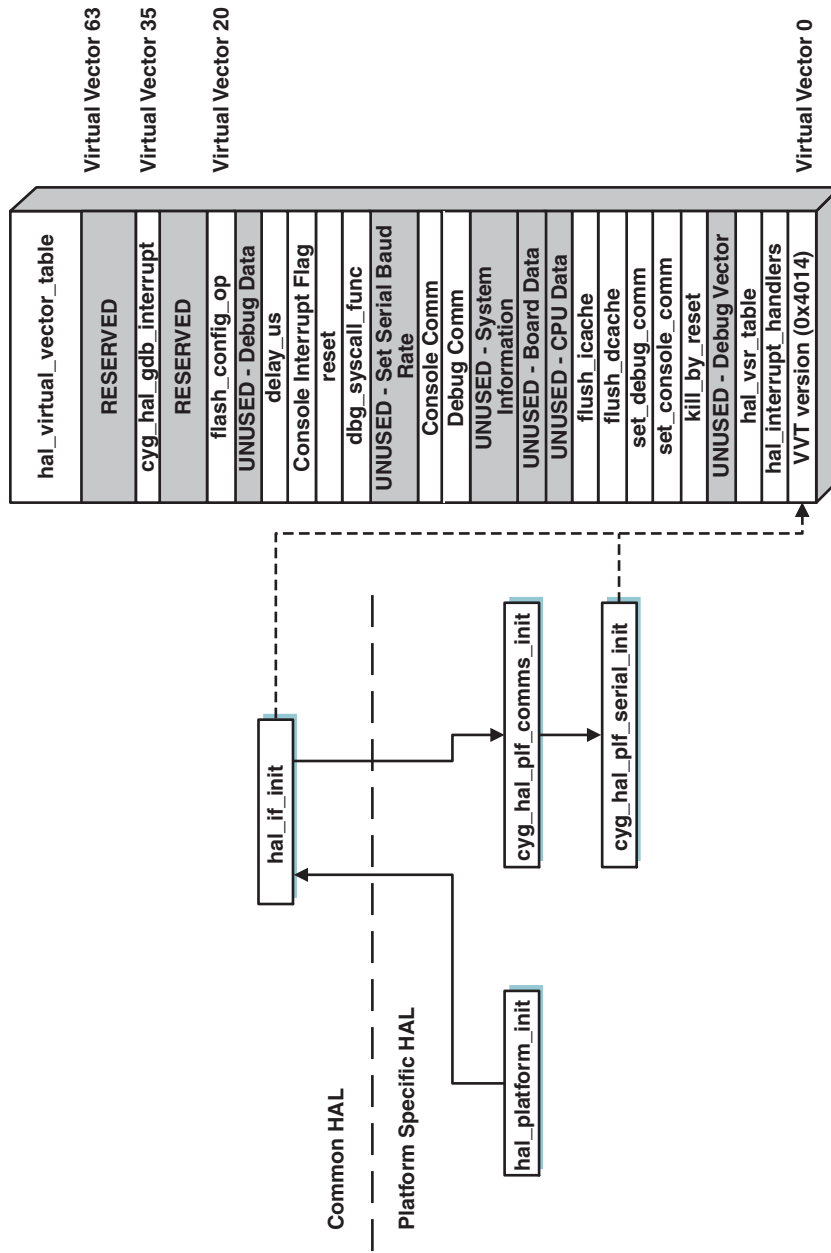


Figure 4.1 Virtual vector table initialization sequence.

We can see in Figure 4.1 that the HAL common function `hal_if_init` initializes the VVT with its default service functions and data. The platform-specific configuration of the VVT is accomplished in the function `cyg_hal_plf_comms_init`, found in the file `hal_diag.c`. The accesses to the VVT are shown with dashed lines in the figure. The platform-specific function initializes the communications channels because the platform code is aware of the number of communications channels supported on the target hardware. During this initialization sequence, the console communication interface tables are allocated and configured with the appropriate HAL platform-specific procedures for accessing the configured console port.

4.1.2.1 Communication Channels

Since the HAL controls the low-level I/O functions for diagnostic and debug communication, it is important to understand the scheme that eCos uses to allow access to the different I/O ports. All HAL I/O happens via the *communication channels*, also called *COMMS* channels. There are two types of COMMS channels within the HAL, console and debug. Each channel type can be individually configured to use any physical port, such as serial or Ethernet, on the target hardware.

Console channels are used for diagnostic I/O during the debugging process; for example, routing `diag_printf` function output for event logging, traces, or assertion messages. Normal I/O communication should use proper device drivers as described in the *I/O Control System* section of Chapter 7. Debug channels are used for communication between the host debugger, such as GDB, and the ROM monitor.

The number of communication channels varies among the different platforms. The configuration option *Number of Communication Channels on the Board* (`CYGNUM_HAL_VIRTUAL_VECTOR_COMM_CHANNELS`), under the HAL architecture-specific components, defines how many channels are present for a particular platform.

There is a *Communication Interface Table* (CIT) associated with each COMMS channel in the system. The CIT, defined as `hal_virtual_comm_table_t` in the common HAL, is an array that contains pointers to procedures or data relevant to the specific COMMS channel. A CIT is allocated, in the file `hal_if.c`, for each COMMS channel supported by the platform using the *Number of Communication Channels on the Board* configuration option, plus an additional CIT for the possible use of mangler procedures. The use of mangler procedures is described later in this section. Table 4.2 lists the supported COMMS CIT procedures. The procedures in the CIT allow access to various COMMS channel functionality.

Table 4.2 Console and Debug Communication Interface Table Procedures

Procedure	Table Index	Description
Channel Data	0	Pointer to the communication controller base address. All procedures in the table use this base address as their first argument.
Write	1	Send a buffer to a device.
Read	2	Get a buffer from a device.

Table 4.2 Console and Debug Communication Interface Table Procedures (Continued)

Procedure	Table Index	Description
Put Character	3	Write a character to a device.
Get Character	4	Read a character from a device.
Control	5	Device settings control. The second argument to this procedure is one of the following functions: Set Baud —Changes the baud rate. Get Baud —Returns the current baud rate. Install Debug ISR —Not used. Remove Debug ISR —Not used. IRQ Disable —Disable debugging receive interrupts. IRQ Enable —Enable debugging receive interrupts. Get Debug ISR Vector —Return the ISR vector for debugging receive interrupts. Set Timeout —Set the Get Character timeout.
Debug ISR	6	ISR used to handle receive interrupts from the device.
Get Character With Timeout	7	Read a character from the device with a timeout.

The console COMMS channel within a RAM application can be configured to use the ROM monitor debug channel or an independent channel. The virtual vector configuration suboption that determines the console COMMS channel used is *Inherit Console Settings From ROM Monitor* (CYGSEM_HAL_VIRTUAL_VECTOR_INHERIT_CONSOLE). This option is enabled by default when using a ROM monitor. If an independent channel is configured by the RAM application, the configuration option *Route Diagnostic Output To Debug Channel* (CYGDBG_HAL_DIAG_TO_DEBUG_CHAN) is available. This configuration option is located under the HAL common configuration components.

NOTE Two scenarios will cause an Ethernet debug connection to be dropped when the application is executed. First, if the console is not inherited by the application using the configuration option *Inherit Console Settings From ROM Monitor* (CYGSEM_HAL_VIRTUAL_VECTOR_INHERIT_CONSOLE). Second, if the debug COMMS channel is reinitialized by the application using the configuration option *Claim COMMS Virtual Vectors* (CYGSEM_HAL_VIRTUAL_VECTOR_CLAIM_COMMS).

To allow diagnostic messages to use the debug COMMS channel, it is necessary to wrap the message with the protocol so that it can be properly displayed by GDB. This wrapping of the message with the protocol is called *mangling*. If the text is not properly mangled, the debugger might reject the message. The HAL provides functions to encapsulate messages according to the selected mangler.

Debuggers, such as GDB, typically use some type of protocol to encode the commands exchanged between the target hardware and the host debugger machine. An explanation of the GDB protocol can be found online at:

http://sources.redhat.com/gdb/onlinedocs/gdb_toc.html

The configuration option *Mangler Used on Diag Output* (CYGSEM_HAL_DIAG_MANGLER), under the HAL common configuration components, allows the selection of a mangler. The possible values for this option are GDB—in which case, the GDB protocol is applied to text messages—or None, which outputs raw text messages. The mangler procedures are contained in a communication interface table supporting the same functionality shown in Table 4.2.

```
1 void cyg_hal_plf_serial_init(  
2     void)  
3 {  
4     hal_virtual_comm_table_t *comm;  
5  
6     // Get the current console communication interface  
7     // table so we can restore it later.  
8     int cur = CYGACC_CALL_IF_SET_CONSOLE_COMM(  
9         CYGNUM_CALL_IF_SET_COMM_ID_QUERY_CURRENT);  
10  
11    // Make sure we only go through this code once.  
12    static int init = 0;  
13    if ( init )  
14        return;  
15  
16    init++;  
17  
18    // Configure the serial port registers  
19    // in the processor.  
20    cyg_hal_plf_serial_init_channel( );  
21  
22    //  
23    // Setup the communication  
24    // interface table routines for  
25    // console I/O for channel 0.  
26    //  
27
```

```
28 // Set the communication
29 // interface table in the VVT to
30 // console 0.
31 CYGACC_CALL_IF_SET_CONSOLE_COMM( 0 );
32
33 // Get a pointer to the channel 0
34 comm = CYGACC_CALL_IF_CONSOLE_PROCS( );
35
36 // Set the base address of the channel 0 controller.
37 CYGACC_COMM_IF_CH_DATA_SET( *comm, eppc_base() );
38
39 // Set the write function.
40 CYGACC_COMM_IF_WRITE_SET(
41     *comm,
42     cyg_hal_plf_serial_write);
43
44 // Set the read function.
45 CYGACC_COMM_IF_READ_SET(
46     *comm,
47     cyg_hal_plf_serial_read);
48
49 // Set the put character function.
50 CYGACC_COMM_IF_PUTC_SET(
51     *comm,
52     cyg_hal_plf_serial_putc);
53
54 // Set the get character function.
55 CYGACC_COMM_IF_GETC_SET(
56     *comm,
57     cyg_hal_plf_serial_getc);
58
59 // Set the serial port control function.
60 CYGACC_COMM_IF_CONTROL_SET(
61     *comm,
62     cyg_hal_plf_serial_control);
63
64 // Set the ISR for debugging receive interrupts.
65 CYGACC_COMM_IF_DBG_ISR_SET(
66     *comm,
67     cyg_hal_plf_serial_isr);
68
69 // Set the get character with timeout function.
70 CYGACC_COMM_IF_GETC_TIMEOUT_SET(
71     *comm,
72     cyg_hal_plf_serial_getc_timeout);
73
```

```
74     // Restore the original
75     // console communication interface
76     // table.
77     CYGACC_CALL_IF_SET_CONSOLE_COMM( cur );
78 }
```

Code Listing 4.2 Motorola PowerPC MBX860 platform communication interface table initialization for console I/O.

Code Listing 4.2 shows the initialization of the console communication interface table for the Motorola PowerPC MBX860 board. The MBX860 board contains a single serial port, designated as channel 0 in the listing. The routine `cyg_hal_plf_serial_init`, shown on line 1, is called from the function `cyg_hal_plf_comms_init` in the platform HAL. The routine `cyg_hal_plf_comms_init` is called from `hal_if_init`, as shown in Figure 4.1.

The first step is to store the current CIT setup in the VVT, as we see on line 8. Next, `cyg_hal_plf_serial_init_channel`, as shown on line 20, is called to configure the appropriate registers in the processor to enable communication via the physical serial port. This function can be called multiple times on other platforms that need to configure more than one serial port.

Next, the call `CYGACC_CALL_IF_SET_CONSOLE_COMM`, on line 31, is made to set the console channel to 0. This puts the CIT allocated for console channel 0 into the VVT, which a pointer to the table is then retrieved by the `CYGACC_CALL_IF_CONSOLE_PROCS` call on line 34. The next eight calls, starting on line 37 and ending on line 72, use the macros `CYGACC_COMM_IF_XXX_SET`, where XXX designates the CIT procedure. These macros set the functions for channel 0 in the CIT. Finally, `CYGACC_CALL_IF_SET_CONSOLE_COMM`, on line 77, is called to restore the CIT in the virtual vector table to its original setting.

The use of the CIT procedures for diagnostic I/O is enabled by the configuration option *Do Diagnostic I/O Via Virtual Vector Table* (`CYGSEM_HAL_VIRTUAL_VECTOR_DIAG`). This allows the console used for diagnostic I/O to be changed during run time.

4.2 Summary

In this chapter, we examined a group of pointers defined in the eCos system called virtual vectors. Virtual vectors and the VVT allow different applications running from different memory to provide varying degrees of functionality. For example, virtual vectors provide a mechanism for a ROM monitor to offer functionality for an application running from RAM. We then looked at the different configuration options for virtual vectors and the different communication channels.

The Kernel

The core of the eCos system is the kernel. We begin this chapter by understanding the functionality provided by the kernel. Next, we cover the startup procedure the kernel follows in order to start an application. Finally, we look at the different scheduling schemes available, how they are used, and how we configure the kernel to meet the needs of different applications.

This chapter, along with the material covered in the next chapter on threads and synchronization, prepares us for building an application that use the eCos kernel and its features.

5.1 The Kernel

The *kernel* is the core to the eCos system. The kernel provides the standard functionality expected in an RTOS, such as interrupt and exception handling, scheduling, threads, and synchronization. These standard functional components that comprise the kernel are fully configurable under the eCos system to meet your specific needs. The eCos kernel is implemented in C++ allowing applications implemented in C++ to interface directly to the kernel; however, there is no official C++ API provided.

There is a configuration option to allow the use of a C kernel API. The kernel API functions are defined in the file `kap_i.h`. The eCos kernel also supports interfacing to standard μ ITRON and POSIX compatibility layers. Further information about the compatibility layers can be found in Chapter 8, *Additional Functionality and Third-Party Contributions*.

A few criteria were the focus of the eCos kernel development to allow it to meet its real-time goals:

- **Interrupt latency**—the time taken to respond to an interrupt and begin execution of an ISR is kept low and deterministic.
- **Dispatch latency**—the time taken from when a thread becomes ready to run to the point it begins execution is kept low and deterministic.
- **Memory footprint**—the memory resources required for both code and data is kept minimal and deterministic for a given system configuration. Dynamic memory allocation is configurable in the core components to ensure that the embedded system does not run out of memory.
- **Deterministic kernel primitives**—the execution of kernel operations is predictable, allowing an embedded system to meet real-time requirements.

The performance measurements of these real-time criteria can be found in the online documentation at:

<http://sources.redhat.com/ecos/docs.html>

The eCos kernel API does not return standard error codes for its functions, a practice typical in most APIs. Error return codes ensure that the application is using the functions correctly. However, in an embedded system, processing error return codes can cause a number of problems such as eating up valuable processing cycles and codes space to check the return value. In addition, in an embedded system, typically there is no way to recover from certain errors so the application would be halted.

Instead, the eCos kernel provides assertions that can be enabled or disabled within the eCos package. Typically, assertions are enabled during debugging, allowing the kernel functions to perform certain error checking. If a problem is discovered, an assertion failure is reported and the application is terminated. This allows you to debug the problem using the various debugging facilities provided. After the debug process is complete, assertions can be disabled in the kernel package. This approach has several advantages such as limiting error checking overhead within the function, eliminating the need for application error checking, and if an error occurs, the application is halted allowing immediate debugging of the problem rather than relying on the check of a return code. Additional information about asserts and tracing can be found in Chapter 7, *Other eCos Architecture Components*.

The kernel components offer methods to ease debugging. One such method, enabled by configuration options, is kernel *instrumentation*. Instrumentation allows the kernel to invoke routines, whenever certain events occur, which write event records into a circular buffer for analysis at a later time. These event records include time stamps, record type, and other supporting data that can be used for kernel debugging or analysis of time-critical kernel events.

5.1.1 Kernel Directory Structure

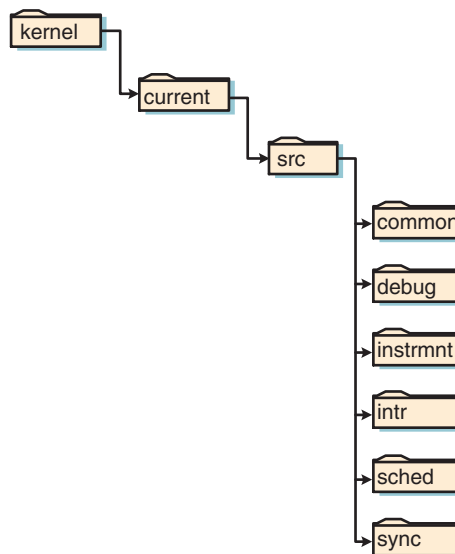
The kernel package is located in the repository under the `kernel` subdirectory. A snapshot of the kernel source file directories, located under `kernel\current\src`, is shown in

Figure 5.1. The `common` subdirectory consists of the implementations for the clock, exception, thread and timer classes, as well as the kernel C API.

The `debug` subdirectory includes the interface calls from a ROM monitor into the kernel, allowing thread-level debugging. The instrumentation code, which allows kernel event logging, is found under the `instrmnt` subdirectory. The `intr` subdirectory contains the kernel interrupt handling class implementation.

Next, the scheduler code, be it bitmap, lottery (which is experimental), or multilevel queue, is in the `sched` subdirectory. Finally, the `sync` subdirectory contains the semaphore, flag, message box, condition variable, and mutex synchronization primitive class implementations.

Figure 5.1 eCos kernel source files directory structure.



5.1.2 Kernel Startup

The kernel startup procedure is invoked from the HAL, after all hardware initialization is complete, as shown in Figure 2.3 in Chapter 2. The last step in Figure 2.3 is to call `cyg_start`, which is the beginning of the kernel startup procedure. The kernel startup procedure is contained in one core function, `cyg_start`, which calls other default startup functions to handle various initialization tasks. These default functions are placeholders for you to override the function with necessary initialization for a specific application. The default kernel startup functions can be overridden by simply providing the same function name in the application code.

NOTE The function `cyg_start` can also be overridden; however, this should rarely be done. The kernel startup procedure provides sufficient override points for the installation of application-specific initialization code.

Code Listing 5.1 shows the prototype functions that must be used to override the different kernel startup routines.

```

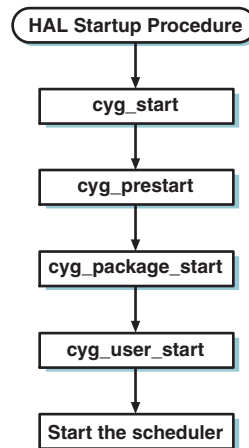
1 void cyg_start( void );
2
3 void cyg_prestart( void );
4
5 void cyg_package_start( void );
6
7 void cyg_user_start( void );

```

Code Listing 5.1 Kernel startup function prototypes.

The kernel startup procedure is shown in Figure 5.2. The core kernel startup function, `cyg_start`, is located in the file `startup.cxx` under the `infra` package subdirectory.

Figure 5.2 Kernel startup procedure.



The next function called from within `cyg_start` is `cyg_prestart`. This default function is located in the file `prestart.cxx` under the `infra` subdirectory. The pre-start function does not perform any initialization tasks. `cyg_prestart` is the first override point in the kernel startup process should any initialization need to be performed prior to other system initialization.

Next, `cyg_package_start` is called. This function is located in the file `pkgstart.cxx` under the `infra` subdirectory. The `cyg_package_start` function allows other packages, such as μ ITRON and ISO C library compatibility, to perform their initialization prior to invoking the application's start function. For example, if the μ ITRON compatibility layer package is configured for use in the system, the function `cyg_uitron_start` is called for initialization. It is possible to override this function if your own package needs initialization; however, you must be sure to invoke the initialization code for the packages included in the configuration.

The function `cyg_user_start` is invoked next. This is the normal application entry point. A default for this function, which does not perform any tasks, is provided in the file `userstart.cxx` under the `infra` subdirectory; therefore, it is not necessary to provide this function in your application. The `cyg_user_start` function is used instead of a `main` function.

NOTE The function `main` can be used as the user application starting point if the ISO C library compatibility package is included in the configuration. To accommodate this, the ISO C library provides a default `cyg_user_start` function that is called if none is supplied by the user application. This default `cyg_user_start` function creates a thread that then calls the user application `main` function.

It is recommended that `cyg_user_start` be used to perform any application-specific initialization, create threads, create synchronization primitives, setting up alarms, and register any necessary interrupt handlers. It is not necessary to invoke the scheduler from the user start function, since this is performed when `cyg_user_start` returns. Code Listing 6.1, found in Chapter 6, is an example of a `cyg_user_start` routine that creates a thread.

The final step in the kernel startup procedure is to invoke the scheduler. The scheduler that is started, either multilevel queue or bitmap, depends on the configuration option settings under the kernel component package.

NOTE Code running during initialization executes with interrupts disabled and the scheduler locked. Enabling interrupts or unlocking the scheduler is not allowed because the system is in an inconsistent state at this point.

Since the scheduler is started after `cyg_user_start` returns, it is important that kernel services are not used within this routine. Initializing kernel primitives, such as a semaphore, is acceptable; however, posting or waiting on a semaphore would cause undefined behavior and possibly system failure.

5.1.3 The Scheduler

The core of the eCos kernel is the *scheduler*. The jobs of the scheduler are to select the appropriate thread for execution, provide mechanisms for these executing threads to synchronize, and control the effect of interrupts on thread execution. This section does not describe the implementation details of the different schedulers, but instead gives a basic understanding of how the existing eCos schedulers operate and the configuration options available.

During the execution of the scheduler code, interrupts are not disabled. Because of this, interrupt latency is kept low.

A counter exists within the scheduler that determines whether the scheduler is free to run or disabled. If the lock counter is nonzero, scheduling is disabled; when the lock counter returns

to zero, scheduling resumes. As described in Chapter 3, the HAL default interrupt handler routine modifies the lock counter to disable rescheduling from taking place during execution of the ISR. Threads also have the ability to lock and unlock the scheduler.

NOTE It is important that the kernel API functions are used to lock and unlock the scheduler, not by accessing the lock variable directly.

On some occasions, it might be necessary for a thread to lock the scheduler in order to access data shared with another thread or DSR. The lock and unlock functions are atomic operations handled by the kernel. Item List 5.1 lists the supported kernel API functions.

Item List 5.1 Kernel Scheduler API Functions

- Syntax: void
 cyg_scheduler_start(
 void
);
- Context: Init
- Parameters: None
- Description: Starts the scheduler, bitmap or multilevel queue, according to the configuration options selected. This call also enables interrupts.
-
- Syntax: void
 cyg_scheduler_lock(
 void
);
- Context: Thread/DSR
- Parameters: None
- Description: Locks the scheduler, preventing any other threads from executing. This function increments the scheduler lock counter.
-
- Syntax: void
 cyg_scheduler_unlock(
 void
);
- Context: Thread/DSR
- Parameters: None
- Description: This function decrements the scheduler lock counter. Threads are allowed to execute when the scheduler lock counter reaches 0.
-
- Syntax: cyg_ucount32
 cyg_scheduler_read_lock(

```
void  
);
```

Context: Thread/DSR
Parameters: None
Description: Returns the current state of the scheduler lock.

eCos supports two different schedulers that implement distinct policies. The eCos kernel is built using only a single scheduler at any one time. The schedulers are:

- Multilevel queue
- Bitmap

NOTE A third scheduler exists in the eCos repository called the lottery scheduler, which is located in the file `lottery.cxx` under the `kernel\current\src\sched` subdirectory. The lottery scheduler is currently an experimental implementation and not shown in any configuration options. Only the multilevel queue and bitmap schedulers are actively supported. To use the lottery scheduler, hand-editing of the eCos system configuration is needed to include the lottery code implementation.

5.1.3.1 Multilevel Queue Scheduler

The *multilevel queue scheduler* allows the execution of multiple threads at each of its priority levels. The number of priority levels is a configuration option from 1 to 32, corresponding to priority numbers 0 (highest priority) to 31 (lowest priority). The scheduler allows preemption between the different priority levels.

Symmetric Multi-Processing (SMP) is only supported when using the multilevel queue scheduler. Additional information about SMP support under eCos can be found in Chapter 8.

Preemption is a context switch halting execution of a lower priority thread, thereby allowing a higher priority thread to execute. The multilevel queue scheduler also allows timeslicing within a priority level.

Timeslicing allows each thread at a given priority to execute for a specified amount of time, which is controlled by a configuration option. The queue implementation for the multilevel scheduler uses double linked circular lists to chain together threads within a priority level and threads at different priority levels.

In Figure 5.3, we see the multilevel scheduling queue representation along with an example of thread execution using this scheduler.

In the scenario shown in Figure 5.3, three threads—Thread A, Thread B, and Thread C—are configured during creation of the threads at priority levels 0, 0, and 30, respectively. The state of the scheduler queue after thread creation is shown in Figure 5.3. For this scenario, timeslicing is enabled. The timeline is a snapshot that starts with Thread C executing.

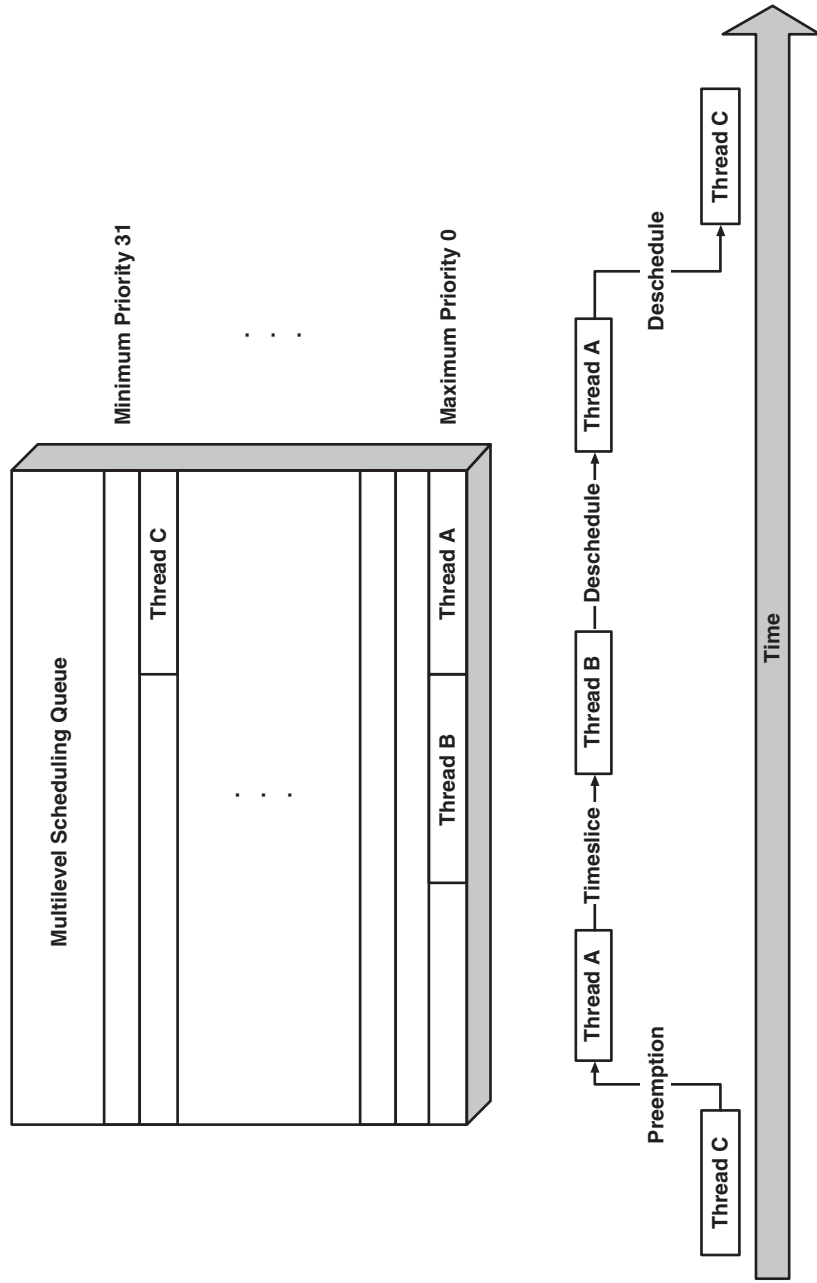


Figure 5.3 Multilevel queue scheduler thread operation.

Next, Thread A becomes able to run, causing Thread C to be preempted and a context switch occurs. During the execution of Thread A, Thread B also becomes able to run. Thread A continues until its timeslice period expires. Then, another context switch occurs allowing Thread B to run. Thread B completes within its given timeslice period. The de-scheduling of a thread can happen for various reasons; for example, by waiting on a mutex that is not free or delaying for a specified amount of time. Since Thread A has the highest priority of tasks waiting to execute, a context switch occurs and it runs next. After Thread A has completed, a context switch takes place allowing Thread C to execute.

5.1.3.2 Bitmap Scheduler

The *bitmap scheduler* allows the execution of threads at multiple priority levels; however, only a single thread can exist at each priority level. This simplifies the scheduling algorithm and makes the bitmap scheduler very efficient. The number of priority levels is a configuration option from 1 to 32, corresponding to priority numbers 0 (highest priority) to 31 (lowest priority).

NOTE When using the bitmap scheduler, it is fatal to set two threads at the same priority number. An assertion is raised if the eCos image is built with assertion support.

The scheduling queue is either an 8-, 16-, or 32-bit value, depending on the number of priority levels selected. A bit in the scheduling queue represents each priority level. The scheduler allows preemption between the different priority levels. Since only one thread is allowed at each priority level, timeslicing is irrelevant and is disabled as a configuration option when using the bitmap scheduler.

Figure 5.4 illustrates an example of thread execution using the bitmap scheduler.

In Figure 5.4, there are three threads created at different priority levels: Thread A—priority 0 (highest), Thread B—priority 1, and Thread C—priority 30 (lowest). The state of the bitmap scheduler queue after the threads are created is shown above the thread execution timeline. The timeline is a snapshot of thread execution starting with Thread C running. Next, Thread A and Thread B are able to run, causing a context switch and Thread C is preempted. Thread A executes next because it has the highest priority of the waiting threads. When Thread A completes, a context switch takes place, enabling Thread B to execute. After Thread B completes, Thread C can finish its processing.

As we can see comparing the execution timelines from Figures 5.3 and 5.4, the bitmap scheduler is a much more simplistic scheduling policy. However, the multilevel queue offers more options for thread operation. The decision of what scheduler to use is dependent on the specific needs of the application.

5.1.3.3 Priority Levels

Both schedulers support thread *priority levels*. The priority level determines which thread will run next of the threads ready to be run. Since the bitmap scheduler only allows a single thread per priority level, the number of priority levels determines the total number of possible threads

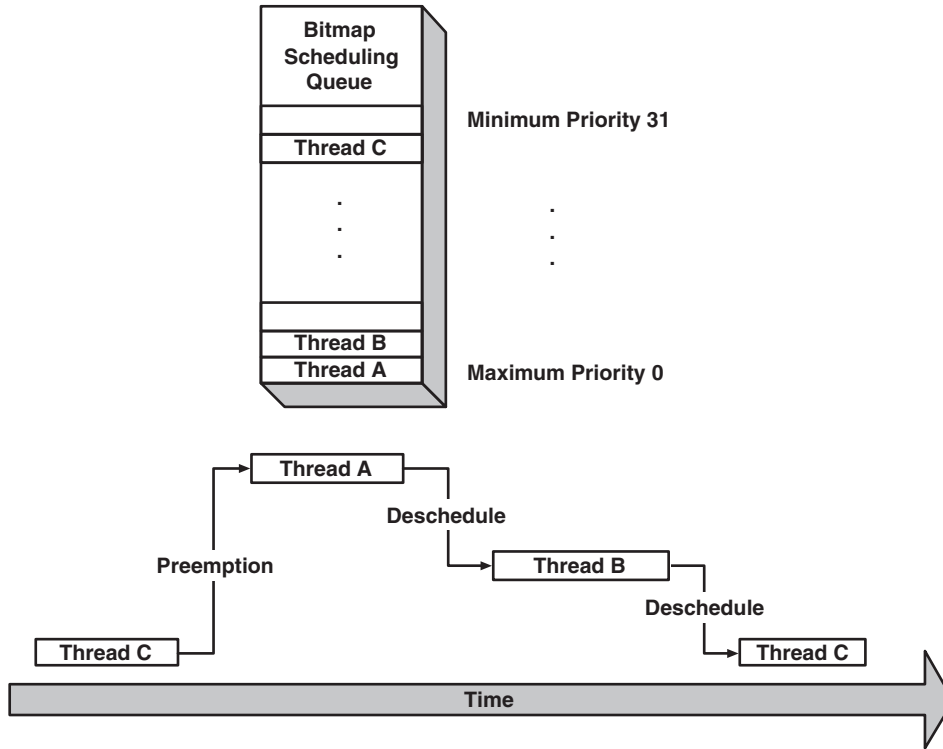


Figure 5.4 Bitmap scheduler thread operation.

in the system. The number of threads possible for the multilevel queue scheduler is independent from the number of priority levels. The maximum number of threads for the multilevel queue scheduler is dependent on the memory resources available.

The maximum number of priority levels allowed is 32. A smaller value for the priority level corresponds the higher the priority of the thread. Item List 5.2 lists the kernel API functions for priority level manipulation of a given thread.

Item List 5.2 Kernel Priority Level API Functions

Syntax: `void
cyg_thread_set_priority(
 cyg_handle_t thread,
 cyg_priority_t priority
);`

Context: Thread

Parameters: `thread`—handle to the thread.
`priority`—level to set the thread priority.

Description: Set the thread to the specified priority level. The valid ranges for the priority value are determined by the configuration option settings. The number of priority levels can be configured from 1 to 32, where lower values represent higher thread priorities.

Syntax:

```
cyg_priority_t
cyg_thread_get_current_priority(
    cyg_handle_t thread
);
```

Context: Thread/DSR

Parameters: *thread*—handle to the thread.

Description: Return the current priority level for the specified thread. This priority value might differ from the priority set for the thread, during creation or by a `cyg_thread_set_priority` function call, since a thread priority boost might have occurred due to the thread using a mutex.

Syntax:

```
cyg_priority_t
cyg_thread_get_priority(
    cyg_handle_t thread
);
```

Context: Thread/DSR

Parameters: *thread*—handle to the thread.

Description: Return the priority for the specified thread. This priority returned is the value last used in a call to `cyg_thread_set_priority` or the value when the thread was created.

5.1.3.4 Scheduler Configuration

The scheduler configuration options are located under the *Kernel Schedulers* component within *eCos Kernel* package. The configuration options allow you to tailor the resources used by the scheduler according to the specific needs of the application. Item List 5.3 details the configuration options available, as well as the different suboptions.

Item List 5.3 Kernel Scheduler Configuration Options

Option Name **Multilevel Queue Scheduler**
CDL Name CYGSEM_KERNEL_SCHED_MLQUEUE
Description Enables the multilevel queue scheduler implementation.

Option Name **Bitmap Scheduler**
CDL Name CYGSEM_KERNEL_SCHED_BITMAP
Description Enables the bitmap scheduler implementation.

Option Name **Number of Priority Levels**
CDL Name CYGNUM_KERNEL_SCHED_PRIORITIES
Description Specifies the number of available priority levels. This number determines the queue size for the specified scheduler. For the bitmap scheduler, this number also determines total number of threads possible. Valid values for this option are 1 to 32, with the default being set to 32. A suboption allows the selection of the de-queue method. When enabled, threads of equal priority are de-queued with the oldest thread first. This suboption is disabled by default.

Option Name	Scheduler Timeslicing
CDL Name	CYGSEM_KERNEL_SCHED_TIMESLICE
Description	Enables timeslicing mode for the multilevel queue scheduler. The scheduler checks to determine if another thread at the same priority level is ready to run. If so, a context switch will take place after the timeslice period, selectable as a suboption in clock ticks between timeslices, expires. This option is enabled by default for the multilevel queue scheduler. Another suboption allows timeslicing to be dynamically enabled or disabled on a per-thread basis.
Option Name	Enable ASR Support
CDL Name	CYGSEM_KERNEL_SCHED_ASR_SUPPORT
Description	Controls Asynchronous Service Routine (ASR) support, which is a function called from the scheduler when it has exited the scheduler lock. This is typically used by compatibility layer packages, such as POSIX.

5.2 Summary

In this chapter, we looked at the functionality provided by the core of eCos, the kernel. We examined the kernel startup procedure. We also looked at the two scheduling algorithms available for the eCos kernel: bitmap and multilevel queue.

Threads and Synchronization Mechanisms

The previous chapter on the eCos kernel provided us with an understanding of how the different schedulers handle threads and their associated priority levels. In this chapter, we start by covering threads; specifically, how we configure the operation of threads, and how we create and use threads in an application.

Building on this, we then cover different mechanisms available for communication between threads and other elements of a typical embedded system. These synchronization mechanisms provide standard tools for developing robust applications.

6.1 Threads

A *thread* is a single flow of execution through a program. Multiple threads can exist in a program, allowing an individual thread to perform its own operations on the system. Each thread defined in the eCos system contains its own context or workspace to perform its operations and a priority level to execute. It is the job of the scheduler to determine which thread is entitled to run at any given time. The kernel contains API functions for controlling threads within an application.

eCos offers configuration options that control the behavior of threads in the system, as well as providing additional features to support various thread operations. Item List 6.1 details the thread configuration options available. These are located under the *Thread-Related Options* component within the *eCos Kernel* package.

Item List 6.1 Kernel Thread Configuration Options

Option Name	Allow Per-Thread Timers
CDL Name	CYGFUN_KERNEL_THREADS_TIMER
Description	Enables clock- and alarm-related functions on a per-thread basis. This option is required when using timed wait operations for semaphores and condition variables. The default for this option is enabled.
Option Name	Support Optional Name For Each Thread
CDL Name	CYGVAR_KERNEL_THREADS_NAME
Description	Allows a string name to be used to identify a thread for debugging purposes. Disabling this option reduces code and data size. The default for this option is enabled.
Option Name	Keep Track of All Threads Using a Linked List
CDL Name	CYGVAR_KERNEL_THREADS_LIST
Description	Enables the kernel to keep a list of threads for easy access when debugging. The default for this option is enabled.
Option Name	Keep Track of the Base of Each Thread's Stack
CDL Name	CYGFUN_KERNEL_THREADS_STACK_LIMIT
Description	Allows the kernel to monitor, and adjust for per-thread data, the lower limit of a thread's stack. This does not perform any type of overflow checking for the stack. The default for this option is enabled.
Option Name	Check Thread Stacks For Overflows
CDL Name	CYGFUN_KERNEL_THREADS_STACK_CHECKING
Description	Causes checks for stack overflowing using signatures at the top and bottom of the thread stacks. This option is enabled when debugging and using asserts. When enabled, suboptions control the amount of checking performed, as well as the size of the signature used for overflow checks. The default for this option is enabled.
Option Name	Measure Stack Usage
CDL Name	CYGFUN_KERNEL_THREADS_STACK_MEASUREMENT
Description	Enabling this option causes a predefined value to be initialized into each thread's stack. The kernel API function <code>cyg_thread_measure_stack_usage</code> can be called to obtain a snapshot of the amount of stack used by the thread. The default for this option is disabled.
Option Name	Support For Per-Thread Data
CDL Name	CYGVAR_KERNEL_THREADS_DATA
Description	Uses an area of memory to store thread-specific data. This data is often used by other packages such as the ISO C library. A suboption defines the number of words to use for the per-thread data, which has a default value of six. The default for this option is enabled.
Option Name	Thread Destructors
CDL Name	CYGPKG_KERNEL_THREADS_DESTRUCTORS
Description	Enables registered destructor functions to be called when a thread exits. When enabled, a suboption controls the number of possible destructors allowed. This has a range of 1 to

65535 with a default value of 8. There is also a suboption that when enabled allows each thread to have its own list of destructors. If this suboption is disabled, there is a global destructor list for all threads.

Option Name	Stack Size For the Idle Thread
CDL Name	CYGNUM_KERNEL_THREADS_IDLE_STACK_SIZE
Description	Specifies the stack size, in bytes, for the idle thread. If a separate interrupt stack is not used, this stack must be able to accommodate the requirements of all interrupt handlers. Valid values for this option are from 512 to 65536 bytes, with a default value of 2048 bytes.
Option Name	Maximal Suspend Count
CDL Name	CYGNUM_KERNEL_MAX_SUSPEND_COUNT_ASSERT
Description	This option aids in debugging by providing an assertion if thread suspends exceed this count value. This option is only used when asserts are included in the configuration. The default for this option is 500.
Option Name	Maximal Wake Count
CDL Name	CYGNUM_KERNEL_MAX_COUNTED_WAKE_COUNT_ASSERT
Description	This option aids in debugging by providing an assertion if thread wakeups exceed this count value. This option is only used when asserts are included in the configuration. The default for this option is 500.
Option Name	Idle Thread Must Always Yield
CDL Name	CYGIMP_IDLE_THREAD_YIELD
Description	If a scheduler configuration contains a single priority level, this option ensures that the idle thread yields to the application threads. The default for this option is enabled.

Item List 6.2 lists the kernel API functions that are available for thread control. For information about the priority-related kernel thread API functions, see Item List 5.2 in Chapter 5.

Item List 6.2 Kernel Thread API Functions

Syntax:	<pre>void cyg_thread_create(cyg_addrword_t sched_info, cyg_thread_entry_t *entry, cyg_addrword_t entry_data, char *name, void *stack_base, cyg_ucount32 stack_size, cyg_handle_t *handle, cyg_thread *thread);</pre>
Parameters:	Init/Thread
Context:	<p>sched_info—scheduler-specific information. For most schedulers, this is the priority value for the thread.</p>

entry—routine that begins execution of the thread.
 entry_data—data value passed to the thread entry routine.
 name—string name of the thread.
 stack_base—base address of the stack for the thread.
 stack_size—size, in bytes, of the stack for the thread.
 handle—returned handle to the thread.
 thread—thread information is stored in the thread memory object pointed to by this parameter.

Description: Creates a thread in a suspended state. It is important to note that a thread will not run until the `cyg_thread_resume` call is made for the thread and the scheduler is started.

NOTE It is up to the caller of this create thread function to ensure that the stack is aligned correctly based on the requirements of the processor.

Syntax: `void
cyg_thread_delay(
 cyg_tick_count_t delay
);`

Parameters: Thread

Context: delay—time for thread to sleep (in ticks).

Description: Places the thread in a sleep state for the specified number of ticks. The number of ticks per second is dependent on the HAL configuration option settings for the real-time clock. For example, a delay value of 1 corresponds to a sleep of 10 milliseconds for a system clock running at 100 Hz. Since, $1 \text{ second} / 100 \text{ Hz} = 0.01 \text{ seconds} = 10 \text{ milliseconds}$.

Syntax: `void
cyg_thread_suspend(
 cyg_handle_t thread
);`

Parameters: Thread/DSR

Context: thread—handle to the thread.

Description: Postpones the execution of a thread. If a thread is suspended multiple times, the resume function call must be made for each suspend function call before the thread will execute.

Syntax: `void
cyg_thread_resume(
 cyg_handle_t thread
);`

Parameters: Thread/DSR

Context: thread—handle to the thread.

Description: Causes a thread to continue execution. This call must be made after a thread is created in order to start the execution of the thread. Each suspend function call must correspond to a resume function call before a thread will continue execution.

Syntax: `void
cyg_thread_yield(
);`

- void
);
- Parameters: Thread
- Context: None
- Description: Gives execution control to the thread that is ready to run at the same priority level. If another thread does not exist, this call has no effect.
- Syntax: void
cyg_thread_kill(
 cyg_handle_t thread
);
- Parameters: Thread
- Context: thread—handle to the thread.
- Description: Causes a thread to exit. This does not release any memory allocated for the thread.
- Syntax: cyg_bool_t
cyg_thread_delete(
 cyg_handle_t thread
);
- Parameters: Thread
- Context: thread—handle to the thread.
- Description: Kills a thread, using the `cyg_thread_kill` function, and removes it from the scheduler. A value of false is returned if the thread cannot be killed. After this call, memory (the thread handle, stack and thread object) created for the thread can be reused. Resources allocated by the thread are not freed by calling this function. In addition, synchronization objects owned by the thread are not unlocked; this is the responsibility of the programmer.
- Syntax: void
cyg_thread_exit(
 void
);
- Parameters: Thread
- Context: None
- Description: Exits the current thread, causing it to be removed from the scheduler.
- Syntax: cyg_bool_t
cyg_thread_add_destructor(
 cyg_thread_destructor_fn fn,
 cyg_addrword_t data
);
- Parameters: Thread. (If the configuration suboption *Per-Thread Destructors* is disabled, this function can be called from Init, Thread or DSR context.)
- Context: fn—destructor function called on thread termination.
data—argument passed to destructor function when it is called.

Description: Register a destructor function that is called when the thread terminates. `TRUE` is returned on success, or `FALSE` on failure. This function can only be called when the configuration option *Thread Destructors* is enabled. If the configuration suboption *Per-Thread Destructors* is enabled, the destructor is registered for the current thread. If this suboption is disabled, the destructor is called for when any thread exits. The configuration suboption *Number of Possible Destructors* sets the maximum value for registered destructors; going over the limit causes a return value of `FALSE`.

Syntax:

```
cyg_bool_t
cyg_thread_rem_destructor(
    cyg_thread_destructor_fn fn,
    cyg_addrword_t data
);
```

Parameters: Thread. (If the configuration suboption *Per-Thread Destructors* is disabled, this function can be called from `Init`, `Thread` or `DSR` context.)

Context: `fn`—destructor function called on thread termination.
`data`—argument passed to destructor function when it is called.

Description: Remove a registered destructor function. `TRUE` is returned on success, or `FALSE` on failure. This function may only be called when the configuration option *Thread Destructors* is enabled.

Syntax:

```
cyg_handle_t
cyg_thread_self(
    void
);
```

Parameters: Thread

Context: None

Description: Returns the handle of the current thread.

Syntax:

```
cyg_addrword_t
cyg_thread_get_stack_base(
    cyg_handle_t thread
);
```

Parameters: Any

Context: `thread`—handle to the thread.

Description: Returns the stack base used during thread creation.

Syntax:

```
cyg_uint32
cyg_thread_get_stack_size(
    cyg_handle_t thread
);
```

Parameters: Any

Context: `thread`—handle to the thread.

Description: Returns the stack size used during thread creation.

Syntax:

```
cyg_uint32
cyg_thread_measure_stack_usage(
```

- ```
 cyg_handle_t thread
);
```
- Parameters: Any
- Context: `thread`—handle to the thread.
- Description: Returns maximum number of bytes of stack spaced used, up to that point, by the specified thread. This function is only available if the configuration *Measure Stack Usage* is enabled.
- Syntax: `cyg_handle_t`  
**`cyg_thread_idle_thread`**(  
 void  
);
- Parameters: Thread/DSR
- Context: None
- Description: Returns the handle to the idle thread, which is created by the kernel.
- Syntax: `void`  
**`cyg_thread_release`**(  
 `cyg_handle_t` thread  
);
- Parameters: Thread/DSR
- Context: `thread`—handle to the thread.
- Description: Break the thread out of any wait, such as a delay or synchronization object wait.
- Syntax: `cyg_ucount32`  
**`cyg_thread_new_data_index`**(  
 void  
);
- Parameters: Init/Thread
- Context: None
- Description: Allocates and returns a new index from the pool for the current thread. This index is used to access the thread-specific data. This requires the option for per-thread data support to be enabled. If no indexes are available, an assertion is raised.
- Syntax: `void`  
**`cyg_thread_free_data_index`**(  
 `cyg_ucount32` index  
);
- Parameters: Init/Thread
- Context: `index`—offset into the per-thread data.
- Description: Returns the per-thread data index to the pool for the current thread. This requires the configuration option *Support For Per-Thread Data* to be enabled.
- Syntax: `CYG_ADDRWORD`  
**`cyg_thread_get_data`**(  
 `cyg_ucount32` index  
);

Parameters: Thread  
 Context: `index`—offset into the per-thread data.  
 Description: Retrieves the per-thread data at the specified index for the current thread. This requires the configuration option *Support For Per-Thread Data* to be enabled.

Syntax: `CYG_ADDRWORD1`  
`cyg_thread_get_data_ptr(`  
`cyg_ucount32 index`  
`);`

Parameters: Thread  
 Context: `index`—offset into the per-thread data.  
 Description: Returns a point to the per-thread data at the specified index for the current thread. This requires the configuration option *Support For Per-Thread Data* to be enabled.

Syntax: `void`  
`cyg_thread_set_data(`  
`cyg_ucount32 index,`  
`CYG_ADDRWORD data`  
`);`

Parameters: Thread  
 Context: `index`—offset into the per-thread data.  
`data`—value to set at per-thread data index.  
 Description: Store data for the current thread at the specified index. This requires the configuration option *Support For Per-Thread Data* to be enabled.

The code in Code Listing 6.1 is an example of how to create a thread using the kernel thread API functions.

```

1 #include <cyg/kernel/kapi.h>
2
3 #define MONITOR_THREAD_STACK_SIZE (2048 / sizeof(int))
4
5 int monitor_thread_stack[MONITOR_THREAD_STACK_SIZE];
6 cyg_handle_t monitor_thread_handle;
7 cyg_thread monitor_thread_obj;
8
9 //
10 // Monitoring thread.
11 //
12 void monitor_thread(cyg_addrword_t index)
13 {
14 unsigned long monitor_counter = 0;
15
16 // Infinite loop for monitor thread.
```

<sup>1</sup> If the configuration suboption *Per-Thread Destructors* is disabled, these functions can be called from *Init*, *Thread* or *DSR* context.

```
17 while (1)
18 {
19 // Delay for 1000 ticks.
20 cyg_thread_delay(1000);
21
22 // Increment the counter.
23 monitor_counter++;
24 }
25 }
26
27 //
28 // Main starting point for the application.
29 //
30 void cyg_user_start(void)
31 {
32 // Create the Monitor thread.
33 cyg_thread_create(
34 12,
35 monitor_thread,
36 0,
37 "Monitor Thread",
38 &monitor_thread_stack,
39 MONITOR_THREAD_STACK_SIZE,
40 &monitor_thread_handle,
41 &monitor_thread_obj);
42
43 // Let the thread run when the scheduler starts.
44 cyg_thread_resume(monitor_thread_handle);
45 }
```

**Code Listing 6.1** Thread initialization example.

In the example shown in Code Listing 6.1, a simple monitor thread, cleverly called `monitor_thread` on line 12, is created in the `cyg_user_start` routine, shown on line 33. The `monitor_thread` waits for 1000 ticks, as we can see on line 20, and then increments a local variable, shown on line 23. It is important to note that threads can be created at anytime during the execution of an application, not just at startup.

As we see in the code, the monitor thread is created with a priority level of 12 (line 34), a stack size of 2048 bytes (lines 3 and 39), and a name “Monitor Thread” (line 37). The application is responsible for providing the stack space, which is `monitor_thread_stack` on line 5, for the thread. The thread stack is defined on line 3 so that the thread stack is aligned properly for the processor.

There is no need for data to be passed into the monitor thread, so the `entry_data` parameter is set to zero when the thread is created, as we see on line 36. The object `monitor_thread_obj` (lines 7 and 41) is where the scheduler will store thread-specific information for the monitor thread.



It is important to note that the kernel API call `cyg_thread_resume`, shown on line 44, is needed if the `monitor_thread` is intended to run when the scheduler starts. The resume thread call takes the handle of the monitor thread, `monitor_thread_handle`, as its parameter. The kernel fills in the thread handle, on line 40, after successful creation of the thread. When the scheduler starts, it places the monitor thread in the ready queue to be executed.

### 6.1.1 Thread Stacks and Stack Sizes

As we learned from the example shown in Code Listing 6.1, the application is responsible for providing the stack for a thread, which is used for local variables and tracking function calls and returns. The stack is typically in the form of static data to eliminate the need for dynamic memory allocation functionality in the kernel.

It is important to understand the size requirements for a given thread so an accurate stack size can be created. This eliminates wasting memory if the stack size is set too large, or, more importantly, avoids overflowing the stack if the stack size is set too small for the thread. Stack overflowing can be a very difficult problem to track down.

The stack size depends on a number of factors, which are determined by the characteristics of the code executed by the thread. For example, numerous nesting function calls or large local arrays might require larger stack sizes for a given thread.

There are also configuration options that can have an affect on a thread's stack usage. The interrupt configuration option *Use Separate Stack For Interrupts*, when disabled allows interrupt handlers to use a thread's stack during execution. The interrupt configuration option *Allow Nested Interrupts* can compound the problem when enabled by allowing higher priority interrupts to occur during execution of another interrupt; therefore, requiring additional stack space. Additional information about the different interrupt configuration options and issues related to a thread's stack are in Item List 3.5 in Chapter 3, *Exceptions and Interrupts*.

The HAL defines two macros that can be used by an application as reasonable sizes for a thread's stack. The values for these macros vary among the different processor architectures. These two macros are typically defined in the architecture HAL in the file `hal_arch.h`, which also gives an explanation of how the values were derived for a particular architecture.

The first macro is `CYGNUM_HAL_STACK_SIZE_MINIMUM`. A thread with this stack size is appropriate for threads that run a single function and make simple system calls. It is illegal to attempt to create a thread with a stack size smaller than this value.

The other macro is `CYGNUM_HAL_STACK_SIZE_TYPICAL`. This value is appropriate for a thread's stack size when nested function execution is limited to approximately six levels with no large arrays on the stack.

The kernel provides code, controlled by configuration options, which can help detect stack overflows. Additional information about these and other kernel thread configuration options can be found in Item List 6.1.

One configuration option is *Check Thread Stacks For Overflows*. When this option is enabled (which is the default case for a debug build), a small amount of space at the stack limit

is filled with a signature. The signature is verified every time a thread context switch occurs. If the signature is not valid, this is an indication that a stack overflow has taken place.

Another configuration option to aid in thread stack size evaluation is *Measure Stack Usage*. When this option is enabled, a thread can call `cyg_thread_measure_stack_usage` to find out the maximum stack used to that point. It is important to realize that this value might not be the absolute maximum since it is possible that no interrupts occurred at the worst possible moment while the thread was executing.

## 6.2 Synchronization Mechanisms

The eCos kernel provides the mechanisms for the threads in the system to communicate with each other and synchronize access to shared resources. The mechanisms provided by the eCos kernel are:

- Mutexes
- Semaphores
- Condition variables
- Flags
- Message boxes
- Spinlocks (for SMP systems)

The kernel also provides API functions that allow applications to make use of the synchronization mechanisms. Some of the synchronization mechanism API functions provided are either blocking or nonblocking.

Blocking function calls, such as `cyg_semaphore_wait`, halt execution of the thread until the API function can complete successfully. Nonblocking function calls, such as `cyg_semaphore_trywait`, attempt to complete successfully; however, if the API function is not successful, a return code indicates the status of the call so the thread can proceed with its execution.

Another type of blocking call, blocking with timeout, also exists for certain synchronization mechanisms. These are API functions that halt execution of the thread for a specified period of time, such as `cyg_semaphore_timed_wait`, while attempting to complete the function call successfully. If the function does not complete successfully before the timeout period, the function returns, indicating an unsuccessful status.

### 6.2.1 Mutexes

The first synchronization mechanism provided by eCos is the mutex. A *mutex* (mutual exclusion object) allows multiple threads to share a resource serially. The resource can be an area of memory or a piece of hardware, such as a Direct Memory Access (DMA) controller.

A mutex is similar to a binary semaphore in that it only has two states—locked and unlocked. However, there are a couple of differences between a binary semaphore and a mutex.

A mutex provides protection against priority inheritance, whereas a binary semaphore does not. Priority inheritance is discussed further later in this section.

A mutex also has the concept of an owner, and only the owner can unlock the mutex. A binary semaphore does not have this requirement; it is possible for one thread to lock a binary semaphore and another thread to unlock it. Once a mutex is locked, it should not be locked again; this might cause undefined behavior. A thread that attempts to lock a mutex that is currently owned by another thread will block until the owner unlocks the mutex.

One issue that arises in real-time systems when using mutexes is *priority inversion*. Priority inversion occurs when a high priority thread is incorrectly prevented from executing by a low priority thread. An example of this is when the high priority thread is waiting on a mutex that is currently owned by the low priority thread. Then, an unrelated medium priority thread preempts the low priority thread, preventing the high priority thread from executing at its proper priority level.

eCos provides two solutions to the priority inversion problem that are selectable as configuration options. The first solution is a *priority ceiling protocol*. In the priority ceiling protocol, all threads that acquire the mutex have their priority level raised to a preconfigured value. This value is available as a configuration option. One disadvantage to this protocol is that the priority level for threads using the mutex must be known ahead of time so the proper ceiling value can be set. Another disadvantage is that if the ceiling value is set too high, other unrelated threads with priority levels below the ceiling can be locked out from executing, possibly causing real-time deadlines to be missed. The priority ceiling protocol is used even when priority inversion is not occurring.

A more elegant solution eCos provides is a *priority inheritance protocol*. The priority inheritance protocol allows a thread that owns the mutex to be raised to the priority level equal to the highest level of all threads waiting for the mutex. The priority inheritance protocol is only used when a higher priority thread is waiting for the mutex. The drawback to using this protocol is that synchronization calls are costlier because the scheduler must comply with the inheritance protocol each time.

The configuration options for the mutex synchronization primitive can be found under the *Synchronization Primitives* component within the *eCos Kernel* package. The main configuration option *Priority Inversion Protection Protocols* (`CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL`) controls the inversion algorithm used for mutex operations. This option enables or disables the use of one of the priority inversion protocols for mutexes. Eliminating the use of a priority inversion protocol reduces code and data sizes. Currently, eCos defines one algorithm for protection against priority inversion called *Simple*, which is only available with the multilevel queue scheduler. The Simple algorithm is designed to be fast and deterministic. The priority protocol used within the Simple algorithm can be set by configuration suboptions. The configuration suboption that specifies the inversion protocol used is *Default Priority Inversion Protocol*, which can be set to `INHERIT`, `CEILING`, or `NONE`. Item List 6.3 lists the configuration suboptions for the mutex priority inversion protocol.

**Item List 6.3** Kernel Mutex Configuration Options

|             |                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Enable Priority Inheritance Protocol</b>                                                                                                                                                                                                                                                                                                                                                                          |
| CDL Name    | CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_INHERIT                                                                                                                                                                                                                                                                                                                                                        |
| Description | Enables the priority inheritance protocol, which causes the owner of a mutex to be executed at the highest priority of all threads waiting for the mutex. The default value for this option is enabled.                                                                                                                                                                                                              |
| Option Name | <b>Enable Priority Ceiling Protocol</b>                                                                                                                                                                                                                                                                                                                                                                              |
| CDL Name    | CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING                                                                                                                                                                                                                                                                                                                                                        |
| Description | Enables the priority ceiling protocol, which causes the owner of a mutex to execute at a preset priority level. The default value for this option is enabled. The suboption Default Priority Ceiling specifies the priority level for the ceiling. The mutex will boost the owner of the thread to this priority level while executing. The default value for this option is 0, which is the maximum priority level. |
| Option Name | <b>No Priority Inversion Protocol</b>                                                                                                                                                                                                                                                                                                                                                                                |
| CDL Name    | CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_NONE                                                                                                                                                                                                                                                                                                                                                           |
| Description | Disables the priority inversion protocol. This option is necessary for the run-time selection of a priority inversion protocol. The default value for this option is enabled.                                                                                                                                                                                                                                        |
| Option Name | <b>Default Priority Inversion Protocol</b>                                                                                                                                                                                                                                                                                                                                                                           |
| CDL Name    | CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT                                                                                                                                                                                                                                                                                                                                                        |
| Description | Defines the default inversion protocol to use when mutexes are created if a protocol is not specified. The possible values for this option are INHERIT, CEILING, or NONE. The default value for this option is INHERIT.                                                                                                                                                                                              |
| Option Name | <b>Specify Mutex Priority Inversion Protocol At Runtime</b>                                                                                                                                                                                                                                                                                                                                                          |
| CDL Name    | CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC                                                                                                                                                                                                                                                                                                                                                        |
| Description | Allows the priority inversion protocol used by a mutex to be specified when the mutex is created. The default value for this option is enabled.                                                                                                                                                                                                                                                                      |

The eCos kernel provides API functions for creating and manipulating mutexes. The mutex functions available in the kernel API are listed in Item List 6.4.

**Item List 6.4** Kernel Mutex API Functions

|              |                                                             |
|--------------|-------------------------------------------------------------|
| Syntax:      | void<br><b>cyg_mutex_init</b> (<br>cyg_mutex_t *mutex<br>); |
| Context:     | Init/Thread                                                 |
| Parameters:  | mutex—pointer to the mutex object to initialize.            |
| Description: | Initialize a mutex in the unlocked state.                   |
| Syntax:      | void<br><b>cyg_mutex_destroy</b> (                          |

```
 cyg_mutex_t *mutex
);
```

Context: Thread

Parameters: *mutex*—pointer to the mutex object to remove.

Description: Destroys a mutex. It is important that the mutex be in the unlocked state when it is destroyed, or else the behavior is undefined.

Syntax: `cyg_bool_t`  
**cyg\_mutex\_lock**(  
 `cyg_mutex_t *mutex`  
);

Context: Thread

Parameters: *mutex*—pointer to the mutex object.

Description: Changes the state of a mutex to the locked state, allowing the thread that called this function to become the owner of the mutex. If the mutex is locked when the thread makes this call, the thread will wait until the mutex is in the unlocked state and then continue to execute. `TRUE` is returned if the mutex has been locked, or `FALSE` if it has not been locked.

Syntax: `cyg_bool_t`  
**cyg\_mutex\_trylock**(  
 `cyg_mutex_t *mutex`  
);

Context: Thread

Parameters: *mutex*—pointer to the mutex object.

Description: Attempts to change the state of a mutex to the locked state. This function call returns immediately. `TRUE` is returned if the mutex has been locked, or `FALSE` if it has not been locked.

Syntax: `void`  
**cyg\_mutex\_unlock**(  
 `cyg_mutex_t *mutex`  
);

Context: Thread

Parameters: *mutex*—pointer to the mutex object.

Description: Allows the owner of a mutex to change the state to unlocked. This function should not be called with a mutex that is currently unlocked.

Syntax: `void`  
**cyg\_mutex\_release**(  
 `cyg_mutex_t *mutex`  
);

Context: Thread

Parameters: *mutex*—pointer to the mutex object.

Description: Releases all threads waiting on the mutex. The threads will return from the call `cyg_mutex_lock` with a value of false and not claim the mutex.

Syntax:       void  
              **cyg\_mutex\_set\_ceiling**(  
                  cyg\_mutex\_t \*mutex,  
                  cyg\_priority\_t priority  
                  );  
Context:       Init/Thread  
Parameters:    mutex—pointer to the mutex object.  
              priority—new priority level for ceiling.  
Description:   Sets the priority level for the ceiling of the specified mutex.

Syntax:       void  
              **cyg\_mutex\_set\_protocol**(  
                  cyg\_mutex\_t \*mutex,  
                  enum cyg\_mutex\_protocol protocol  
                  );  
Context:       Init/Thread  
Parameters:    mutex—pointer to the mutex object.  
              protocol—protocol to use for mutex. Valid values are CYG\_MUTEX\_NONE,  
              CYG\_MUTEX\_INHERIT, or CYG\_MUTEX\_CEILING.  
Description:   Sets the protocol for the specified mutex.

Code Listing 6.2 is an example showing two threads using a mutex. The thread and mutex initializations are left out in this example to focus on the use of the mutex.

```
1 #include <cyg/kernel/kapi.h>
2 #include <cyg/io/io.h>
3
4 cyg_io_handle_t port_handle;
5 cyg_mutex_t mut_shared_port;
6
7 //
8 // Thread A.
9 //
10 void thread_a(cyg_addrword_t index)
11 {
12 int result;
13 int write_length = 6;
14 unsigned char write_buffer[6] = {4,21,4,20,6,28};
15
16 // Run this thread forever.
17 while (1)
18 {
19 // Get the mutex.
20 cyg_mutex_lock(&mut_shared_port);
21
22 // Write data to the I/O port.
23 result = cyg_io_write(port_handle,
```

```
24 &write_buffer[0],
25 &write_length);
26
27 // Release the mutex.
28 cyg_mutex_unlock(&mut_shared_port);
29
30 // Get more data to send to the port...
31 }
32 }
33
34 //
35 // Thread B.
36 //
37 void thread_b(cyg_addrword_t index)
38 {
39 int result;
40 int read_length = 3;
41 unsigned char read_buffer[3];
42
43 // Run this thread forever.
44 while (1)
45 {
46 // Get the mutex.
47 cyg_mutex_lock(&mut_shared_port);
48
49 // Read data from the I/O port.
50 result = cyg_io_read(port_handle,
51 &read_buffer[0],
52 &read_length);
53
54 // Release the mutex.
55 cyg_mutex_unlock(&mut_shared_port);
56
57 // Process the data read from the port...
58 }
59 }
```

**Code Listing 6.2** Mutex example code.

As we can see in Code Listing 6.2, Thread A and Thread B both use the same hardware I/O port. The `mut_shared_port` mutex protects the port so that only one thread accesses the port at a time. In this example, we assume Thread A acquires the mutex first by executing its `cyg_mutex_lock` function call on line 20. Thread A is then able to write out its data to the I/O port on line 23. While Thread A is writing out to the port, Thread B executes. However, Thread B must wait when it reaches its `cyg_mutex_lock` function call, shown on line 47, since the mutex is already owned by Thread A.

After Thread A finishes writing out its data, the function call `cyg_mutex_unlock`, on line 28, releases the mutex. Then, Thread B becomes the owner of the mutex and is allowed to access the port. Finally, after Thread B reads its data from the I/O port, on line 50, it releases the mutex with the call `cyg_mutex_unlock` on line 55.

## 6.2.2 Semaphores

A *semaphore* is a synchronization mechanism that contains a count indicating whether a resource is locked or available. There are two types of semaphores, counting and binary. Binary semaphores are similar to counting semaphores; however, their count is never incremented past a value of one. Binary semaphores are in either a locked or unlocked state.

Counting semaphores can be in multiple states depending on their count value. Counting semaphore objects contain a value that is incremented when a thread posts to a semaphore, and the value is decremented when a thread completes a wait for the semaphore. Only the highest priority waiting thread is executed when the semaphore count is above zero. Counting semaphores are often used when a higher priority thread or DSR, which received data, needs to signal another thread to continue processing the data at a lower priority.

The eCos kernel provides API functions for creating and manipulating semaphores. The kernel API is for counting semaphores and not binary semaphores. These API functions, which are defined in Item List 6.5, use counting semaphores.

### Item List 6.5 Kernel Semaphore API Functions

Syntax:       void  
              **cyg\_semaphore\_init**(  
              cyg\_sem\_t \*sem,  
              cyg\_ucount32 val  
              );

Context:       Init/Thread

Parameters:   sem—pointer to semaphore object.  
              val—initial count for semaphore.

Description:   Initializes a semaphore with a count value specified in the val parameter.

Syntax:       void  
              **cyg\_semaphore\_destroy**(  
              cyg\_sem\_t \*sem  
              );

Context:       Thread

Parameters:   sem—pointer to semaphore object.

Description:   Destroys a semaphore. It is important that there are not any threads waiting on the semaphore when this function is called or the behavior is undefined.

Syntax:       void  
              **cyg\_semaphore\_wait**(



|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | <pre>    cyg_sem_t *sem     );</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Context:     | Thread                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Parameters:  | <code>sem</code> —pointer to semaphore object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description: | When the semaphore count is zero, the thread calling this function will wait for the semaphore. When the semaphore count is nonzero, the count will be decremented and the thread calling this function will continue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Syntax:      | <pre>cyg_bool_t <b>cyg_semaphore_trywait</b>(     cyg_sem_t *sem     );</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Context:     | Thread/DSR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Parameters:  | <code>sem</code> —pointer to semaphore object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description: | Attempts to decrement the semaphore count. If the semaphore count is greater than zero, the count is decremented and <code>TRUE</code> is returned. If the count is zero, the semaphore is unchanged and <code>FALSE</code> is returned. In either case, the thread does not block waiting for the semaphore.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Syntax:      | <pre>cyg_bool_t <b>cyg_semaphore_timed_wait</b>(     cyg_sem_t *sem,     cyg_tick_count_t abstime     );</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Context:     | Thread                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Parameters:  | <code>sem</code> —pointer to semaphore object.<br><code>abstime</code> —absolute time to wait for semaphore.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description: | Attempts to decrement a semaphore count. This function is only available when the <i>Allow Per-Thread Timers</i> configuration option is enabled. If the semaphore count is greater than zero, the count is decremented and <code>TRUE</code> is returned. If the count is zero, the function call will wait for the amount of time specified in the <code>abstime</code> parameter. If the timeout occurs before the semaphore count can be decremented, <code>FALSE</code> is returned and the current thread will continue to run. The <code>abstime</code> parameter is an absolute time measured in clock ticks. The following shows how to use a relative wait time:<br><pre>cyg_semaphore_timed_wait(     &amp;sem,     cyg_current_time( ) + 100);</pre><br>In this example, the thread will wait for the semaphore for 100 ticks from the present time. |
| Syntax:      | <pre>void <b>cyg_semaphore_post</b>(     cyg_sem_t *sem     );</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Context:     | Thread/DSR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Parameters:  | <code>sem</code> —pointer to semaphore object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description: | Increment the semaphore count. If a thread is waiting on the specified semaphore, it is awakened.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

Syntax:       void  
              **cyg\_semaphore\_peek**(  
              cyg\_sem\_t \*sem,  
              cyg\_count32 \*val  
              );

Context:       Thread/DSR

Parameters:   sem—pointer to semaphore object.  
              val—pointer to returned semaphore count.

Description:   Returns the current semaphore count in the variable pointed to by the parameter val.

Code Listing 6.3 is a simple example of the creation of a semaphore for use by two threads.

```
1 #include <cyg/kernel/kapi.h>
2 #include <cyg/infra/diag.h>
3
4 #define THREAD_A_STACK_SIZE (2048 / sizeof(int))
5 #define THREAD_B_STACK_SIZE (2048 / sizeof(int))
6
7 cyg_sem_t sem_get_data;
8 int thread_a_stack[THREAD_A_STACK_SIZE];
9 int thread_b_stack[THREAD_B_STACK_SIZE];
10 cyg_handle_t thread_a_handle;
11 cyg_handle_t thread_b_handle;
12 cyg_thread thread_a_obj;
13 cyg_thread thread_b_obj;
14
15 //
16 // Thread A.
17 //
18 void thread_a(cyg_addrword_t index)
19 {
20 // Run this thread forever.
21 while (1)
22 {
23 // Delay for 1000 ticks.
24 cyg_thread_delay(1000);
25
26 // Display a message.
27 diag_printf("Thread A: Signal Thread B!\n");
28
29 // Signal Thread B to run.
30 cyg_semaphore_post(&sem_get_data);
31 }
32 }
33
34 //
35 // Thread B.
```

```
36 //
37 void thread_b(cyg_addrword_t index)
38 {
39 // Run this thread forever.
40 while (1)
41 {
42 // Signal Thread B to run.
43 cyg_semaphore_wait(&sem_get_data);
44
45 // Display a message.
46 diag_printf("Thread B: Got the signal!\n");
47 }
48 }
49
50 //
51 // Main starting point for the application.
52 //
53 void cyg_user_start(
54 void)
55 {
56 // Initialize the get data semaphore to 0.
57 cyg_semaphore_init(&sem_get_data, 0);
58
59 // Create Thread A.
60 cyg_thread_create(
61 12,
62 thread_a,
63 0,
64 "Thread A",
65 &thread_a_stack,
66 THREAD_A_STACK_SIZE,
67 &thread_a_handle,
68 &thread_a_obj);
69
70 // Create Thread B.
71 cyg_thread_create(
72 12,
73 thread_b,
74 0,
75 "Thread B",
76 &thread_b_stack,
77 THREAD_B_STACK_SIZE,
78 &thread_b_handle,
79 &thread_b_obj);
80
81 // Let the threads run when the scheduler starts.
```

```
82 cyg_thread_resume(thread_a_handle);
83 cyg_thread_resume(thread_b_handle);
84 }
```

**Code Listing 6.3** Semaphore example code.

In Item List 6.3, we can see in the function `cyg_user_start`, on line 53, that the `sem_get_data` semaphore is initialized, shown on line 57.

---

**NOTE** This might be understood; however, it should be pointed out nonetheless. It is important to initialize any semaphores, and any other synchronization mechanisms, prior to creating and resuming the threads that use these mechanisms. Undefined behavior will result if this rule is not followed. Careful consideration should also be given to the initial values of synchronization mechanisms to ensure that threads are running at the proper time.

When the semaphore is initialized, a parameter is passed in that determines the initial value of the semaphore count; we can see on line 57 that this value is 0. If the count value is initialized to zero, all threads waiting on the semaphore will continue to wait until a post, which increments the count value, to the semaphore occurs. If the count value is initialized to a value greater than zero, the scheduler will determine which waiting thread to run based on each thread's priority level until the semaphore count value is zero. Each time a wait function succeeds, the semaphore count value is decremented by one.

Thread A executes a delay of 1000 ticks (line 24), outputs a message (line 27), and then posts to the semaphore (line 30). The scheduler then wakes up Thread B because of the semaphore wait call on line 43, outputs a message (line 46), and then returns to the waiting state (line 43).

### 6.2.3 Condition Variables

Another available synchronization mechanism is the *condition variable*. Condition variables are used with mutexes that allow multiple threads access to shared data. Typically, there is a single thread producing the data, and one or more threads waiting for the data to be available. The thread producing the data can either signal a single thread to wake up or all threads to wake up, with a broadcast signal, when the data is available. The waiting threads can then process the data as needed. Item List 6.6 lists the kernel API condition variable control functions.

eCos contains two configuration options for condition variables. These are located in the *Synchronization Primitives* component within the *eCos Kernel* package. The first configuration option is *Condition Variable Timed-Wait Support* (`CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT`), which allows the `cyg_cond_timed_wait` kernel API function to be used by applications. This option is enabled by default.

The second configuration option is *Condition Variable Explicit Mutex Wait Support* (`CYGMFN_KERNEL_SYNCH_CONDVAR_WAIT_MUTEX`), which permits a thread to provide a

different mutex in a call to the wait functions. In the default case, condition variables are created with a statically associated mutex. This configuration option is enabled by default.

#### Item List 6.6 Condition Variable API Functions

- Syntax: `void  
cyg_cond_init(  
    cyg_cond_t *cond,  
    cyg_mutex_t *mutex  
);`
- Context: Init/Thread
- Parameters: `cond`—pointer to condition variable object.  
`mutex`—pointer to mutex object to attach to condition variable.
- Description: Initializes a condition variable and attaches it to the specified mutex.
- 
- Syntax: `void  
cyg_cond_destroy(  
    cyg_cond_t *cond  
);`
- Context: Init/Thread
- Parameters: `cond`—pointer to condition variable object.
- Description: Destroys a condition variable. It is important that this function not be called with a condition variable that is in use.
- 
- Syntax: `cyg_bool_t  
cyg_cond_wait(  
    cyg_cond_t *cond  
);`
- Context: Thread
- Parameters: `cond`—pointer to condition variable object.
- Description: Causes the current thread to wait on the specified condition variable and simultaneously unlocks the mutex attached to the condition variable. Returns `TRUE` on success, and `FALSE` if the thread is forcibly awakened. This call must be made while the thread has the mutex attached to the condition variable locked. The condition variable signal or broadcast functions wake up threads calling this wait function. When the thread wakes up, the mutex is reclaimed prior to this function proceeding. Since the thread might have to wait for the mutex, this wait function should be called within a loop because the condition might become `FALSE` during the waiting period.
- 
- Syntax: `cyg_bool_t  
cyg_cond_timed_wait(  
    cyg_cond_t *cond,  
    cyg_tick_count_t abstime  
);`
- Context: Thread
- Parameters: `cond`—pointer to condition variable object.  
`abstime`—absolute time to wait for condition variable.

**Description:** Causes the thread to wait on the specified condition variable for an absolute time period. This function is only available when the *Condition Variable Timed-Wait Support* configuration option is enabled. If a condition variable signal or broadcast is received, this function returns TRUE. If the timeout occurs before the condition variable is signaled or the thread is forcibly awakened, FALSE is returned. The `abstime` parameter is an absolute time measured in clock ticks. The following shows how to use a relative wait time:

```
cyg_cond_timed_wait(
 &cond,
 cyg_current_time() + 100);
```

In this example, the thread will wait for the condition variable for 100 ticks from the present time.

**Syntax:**

```
void
cyg_cond_signal(
 cyg_cond_t *cond
);
```

**Context:** Thread/DSR

**Parameters:** `cond`—pointer to condition variable object.

**Description:** Wake up exactly one thread waiting on the condition variable, causing that thread to become the owner of the mutex. It is important to understand that a race condition could arise if more than one thread is waiting for the condition variable. This is why it is important for the waiting thread to retest the condition variable to ensure its proper state. If there are no threads waiting for the condition variable when it is signaled, nothing happens. The state of the condition variable is not stored anywhere. Therefore, the next thread that waits on the condition variable needs to wait until the variable is signaled again. If there were a need for the signaled state to be stored, a semaphore would be a better synchronization mechanism to use.

**Syntax:**

```
void
cyg_cond_broadcast(
 cyg_cond_t *cond
);
```

**Context:** Thread/DSR

**Parameters:** `cond`—pointer to condition variable object.

**Description:** Wake up all threads waiting on the condition variable. Each thread that was waiting on the condition variable becomes the owner of the mutex when it runs.

Code Listing 6.4 shows an example using a condition variable. The thread, condition variable, and mutex initializations are left out in this example to focus on the use of the condition variable.

```
1 #include <cyg/kernel/kapi.h>
2 #include <cyg/infra/cyg_type.h>
3
4 unsigned char buffer_empty = true;
5 cyg_mutex_t mut_cond_var;
6 cyg_cond_t cond_var;
```

```
7
8 //
9 // Thread A.
10 //
11 void thread_a(cyg_addrword_t index)
12 {
13 // Run this thread forever.
14 while (1)
15 {
16 // Acquire data into the buffer...
17
18 // There is data in the buffer now.
19 buffer_empty = false;
20
21 // Get the mutex.
22 cyg_mutex_lock(&mut_cond_var);
23
24 // Signal the condition variable.
25 cyg_cond_signal(&cond_var);
26
27 // Release the mutex.
28 cyg_mutex_unlock(&mut_cond_var);
29 }
30 }
31
32 //
33 // Thread B.
34 //
35 void thread_b(cyg_addrword_t index)
36 {
37 // Run this thread forever.
38 while (1)
39 {
40 // Get the mutex.
41 cyg_mutex_lock(&mut_cond_var);
42
43 // Wait for the data and the condition variable signal.
44 while (buffer_empty == true)
45 {
46 cyg_cond_wait(&cond_var);
47 }
48
49 // Get the buffer data...
50
51 // The data in the buffer has been processed.
52 buffer_empty = true;
53
54 // Release the mutex.
```

```
55 cyg_mutex_unlock(&mut_cond_var);
56
57 // Process the data in the buffer...
58 }
59 }
```

**Code Listing 6.4** Condition variable example code.

In Code Listing 6.4, Thread A is acquiring data that is processed by Thread B. First, Thread B executes. On line 41, Thread B acquires the mutex associated with the condition variable. Next, since there is no data in the buffer to process, and `buffer_empty` is `true` on initialization (line 4), Thread B calls `cyg_cond_wait` on line 46. This call to `cyg_cond_wait` does two things—first, it suspends Thread B waiting for the condition variable to be set, and second, it unlocks the mutex `mut_cond_var`.

Now, an event occurs causing Thread A to execute and acquire data into a buffer, as we see on line 16. Next, `buffer_empty` is set to `false` on line 19. Thread A then locks the mutex (line 22), signals the condition variable (line 25), and then unlocks the mutex (line 28).

Next, Thread B is able to run because the condition variable is signaled from Thread A. Before returning from `cyg_cond_wait`, the mutex, `mut_cond_var`, is locked and owned by Thread B. Now, Thread B can get the data buffer (line 49) and set the `buffer_empty` flag to `true` (line 52). Finally, the mutex is released by Thread B on line 55 and the data in the buffer is processed, as we see on line 57.

It is important to understand a couple of issues relating to the code in Code Listing 6.4. First, the mutex unlock and wait code execution in the call to `cyg_cond_wait`, on line 46, is atomic; therefore, no other thread is allowed to run between the unlock and the wait. If this code were not atomic, then it would be possible for Thread B to miss the signal call from Thread A even though data was in the buffer. Why? Because Thread B calls `cyg_cond_wait`, which first checks to see if the condition variable is set; in this case, it is not. Next, the mutex is released in the `cyg_cond_wait` call. Now, Thread A executes, putting data into the buffer and then signaling the condition variable (line 25). Then, Thread B returns to waiting, however, the condition variable has been set.

Another issue to keep in mind is that the call to `cyg_cond_wait` by Thread B is in a `while` loop, on lines 44 through 47. This ensures that the condition that Thread B is waiting on is still true after returning from the condition wait call. Take the case where other threads are waiting on the same condition. Another thread might be queued to obtain the mutex before Thread B; therefore, being signaled and waking up before Thread B. When Thread B finally gets to run, the condition is then false. The `while` loop around the condition wait ensures the condition is still true before a thread executes.



## 6.2.4 Flags

*Flags* are synchronization mechanisms represented by a 32-bit word. Each bit in the flag represents a condition, which allows a thread to wait for either a single condition or a combination of conditions. The waiting thread specifies if all conditions or a combination of conditions are to be met before it wakes up. The signaling thread can then set or reset bits according to specific conditions so the appropriate thread can be executed. The kernel API functions for creating and controlling the flags are detailed in Item List 6.7.

### Item List 6.7 Kernel Flag API Functions

Syntax:       void  
              **cyg\_flag\_init**(  
              cyg\_flag\_t \*flag  
              );

Context:       Init/Thread  
Parameters:   flag—pointer to flag object.  
Description:   Initializes a flag variable.

Syntax:       void  
              **cyg\_flag\_destroy**(  
              cyg\_flag\_t \*flag  
              );

Context:       Init/Thread  
Parameters:   flag—pointer to flag object.  
Description:   Destroys the specified flag variable. Flag variables that are being waited on must not be destroyed.

Syntax:       void  
              **cyg\_flag\_setbits**(  
              cyg\_flag\_t \*flag,  
              cyg\_flag\_value\_t value  
              );

Context:       Thread/DSR  
Parameters:   flag—pointer to flag object.  
              value—the bits that are set to one in this parameter are set along with the current bits set in the flag.  
Description:   Sets the bits in the flag value that are set to one in the `value` parameter. Any threads that are waiting on the flag and have their conditions met are woken up.

Syntax:       void  
              **cyg\_flag\_maskbits**(  
              cyg\_flag\_t \*flag,  
              cyg\_flag\_value\_t value  
              );

Context:       Thread/DSR

- Parameters: `flag`—pointer to flag object.  
`value`—the bits that are set to zero in this parameter are cleared in the flag.
- Description: Clears the bits in the flag value that are set to zero in the `value` parameter. No threads are awakened by this call.
- Syntax: `cyg_flag_value_t`  
**`cyg_flag_wait`**(  
    `cyg_flag_t *flag`,  
    `cyg_flag_value_t pattern`,  
    `cyg_flag_mode_t mode`  
);
- Context: Thread
- Parameters: `flag`—pointer to flag object.  
`pattern`—bit setting that will cause the calling thread to be woken up.  
`mode`—specifies the conditions for wake up.
- Description: If the `mode` parameter is set to AND, the function will wait for all bits in the `pattern` parameter to be set in the flag value. If the `mode` parameter is set to OR, the function will wait for any bits in the `pattern` parameter to be set in the flag value. When this function call returns, the condition is met and the flag value is returned. Zero is returned if the thread is forcibly woken up or forced to exit with `cyg_thread_exit`. The `mode` parameter can have the following possible values:  
`CYG_FLAG_WAITMODE_AND`—wake up if all bits specified in the mask are set in the flag.  
`CYG_FLAG_WAITMODE_OR`—wake up if any bits specified in the mask are set in the flag.  
`CYG_FLAG_WAITMODE_CLR`—clear all bits in the flag when the condition is met. Typically, only the bits that are set in the `pattern` parameter are cleared. This flag is bitwise combined with either the AND or OR wait mode flags.
- Syntax: `cyg_flag_value_t`  
**`cyg_flag_timed_wait`**(  
    `cyg_flag_t *flag`,  
    `cyg_flag_value_t pattern`,  
    `cyg_flag_mode_t mode`,  
    `cyg_tick_count_t abstime`  
);
- Context: Thread
- Parameters: `flag`—pointer to flag object.  
`pattern`—bit setting that will cause the calling thread to be woken up.  
`mode`—modifies the conditions for wake up.  
`abstime`—absolute time to wait for flag conditions to be met.
- Description: Waits for the conditions required by the `pattern` and `mode` parameters or the timeout specified by the `abstime` parameter. If the timeout occurs before the conditions are met, zero is returned; otherwise, the flag value is returned. This function is only available when the *Allow Per-Thread Timers* configuration option is enabled.
- Syntax: `cyg_flag_value_t`  
**`cyg_flag_poll`**(  
    `cyg_flag_t *flag`,

```

 cyg_flag_value_t pattern,
 cyg_flag_mode_t mode
);

```

**Context:** Thread/DSR

**Parameters:** `flag`—pointer to flag object.  
`pattern`—bit setting that will cause the calling thread to be woken up.  
`mode`—modifies the conditions for returning the flag value.

**Description:** If the conditions required by the `pattern` and `mode` parameters are met, the flag value is returned. If these conditions are not met, zero is returned; otherwise, the flag value is returned. Specifying `CYG_FLAG_WAITMODE_CLR` in the `mode` parameter will clear the flag value to zero.

**Syntax:**

```

 cyg_flag_value_t
 cyg_flag_peek(
 cyg_flag_t *flag
);

```

**Context:** Thread/DSR

**Parameters:** `flag`—pointer to flag object.

**Description:** Returns the current value of the specified flag.

**Syntax:**

```

 cyg_bool_t
 cyg_flag_waiting(
 cyg_flag_t *flag
);

```

**Context:** Thread/DSR

**Parameters:** `flag`—pointer to flag object.

**Description:** Returns TRUE if there are any threads waiting on the specified flag.

Code Listing 6.5 shows an example using the kernel API for flags. The thread and flag initializations are left out in this example to focus on the use of flags.

```

1 #include <cyg/kernel/kapi.h>
2
3 cyg_flag_t flag_var;
4
5 //
6 // Thread A.
7 //
8 void thread_a(cyg_addrword_t index)
9 {
10 // Run this thread forever.
11 while (1)
12 {
13 // Delay for 1000 ticks.
14 cyg_thread_delay(1000);
15
16 // Set the appropriate flag bits to signal Thread B.

```

```
17 cyg_flag_setbits(&flag_var, 1);
18 }
19 }
20
21 //
22 // Thread B.
23 //
24 void thread_b(cyg_addrword_t index)
25 {
26 // Run this thread forever.
27 while (1)
28 {
29 // Wait for the appropriate bits to be set in the flag.
30 cyg_flag_wait(&flag_var,
31 3,
32 CYG_FLAG_WAITMODE_OR |
33 CYG_FLAG_WAITMODE_CLR
34);
35 }
36 }
```

**Code Listing 6.5** Flags example code.

Code Listing 6.5 shows a basic example of how Thread A is using the flag, `flag_var` declared on line 3, to signal Thread B. Thread B waits on the `flag_var` flag using the `cyg_flag_wait` function call, as shown on line 30. The second parameter, on line 31, determines the bit pattern, in this case 3, that the flag variable needs to be set to in order to wake up Thread B. The mode parameters, on lines 32 and 33, specify the conditions for wake up. In this case, `CYG_FLAG_WAITMODE_OR` means that Thread B will wake up if either bit 1 or bit 2 is set in the flag. The mode parameter `CYG_FLAG_WAITMODE_CLR` indicates that all bits in the flag are cleared when the condition is met. Thread A sets bit 1 in `flag_var` using the function call `cyg_flag_setbits`, as shown on line 17. Since Thread B is waiting on either bit 1 or bit 2 to be set, Thread B is then awakened.

### 6.2.5 Message Boxes

Another synchronization mechanism provided by eCos are *message boxes*, also called *mailboxes*. Message boxes provide a means for two threads to exchange information. Typically, one thread will produce messages and send to another thread for processing. Message boxes offer another method for threads to communicate more than a single byte of information. Item List 6.8 describes the kernel API message box functions.

There are two configuration options for the message box synchronization mechanism. These are located under the *Synchronization Primitives* component within the *eCos Kernel* package. The first configuration option is *Message Box Blocking Put Support* (`CYGMFN_KERNEL_SYNC_`

MBOXT\_PUT\_CAN\_WAIT). This option, which is enabled by default, allows the put and timed put function calls to be used when sending messages.

The second configuration option determines the number of messages that can be queued in a *Message Box Queue Size* (CYGNUM\_KERNEL\_SYNCH\_MBOX\_QUEUE\_SIZE). The valid values for this option are 1 to 65535 with a default value of 10 messages.

### Item List 6.8 Kernel Message Box API Functions

- Syntax:       void  
               **cyg\_mbox\_create** (  
                   cyg\_handle\_t \*handle,  
                   cyg\_mbox \*mbox  
               );
- Context:       Init/Thread
- Parameters:   handle—pointer to the handle for the new message box.  
               mbox—pointer to new message box object.
- Description:   Constructs a message box in the space pointed to by the `mbox` parameter.
- 
- Syntax:       void  
               **cyg\_mbox\_delete** (  
                   cyg\_handle\_t mbox  
               );
- Context:       Thread
- Parameters:   mbox—handle to the message box.
- Description:   Removes the specified message box. You must not call this function if there is an outstanding get operation. The contents of the message box are not cleaned up if the message box is not empty.
- 
- Syntax:       void\*  
               **cyg\_mbox\_get** (  
                   cyg\_handle\_t mbox  
               );
- Context:       Thread
- Parameters:   mbox—handle to the message box.
- Description:   Removes a message from the specified message box when it is available and returns the address of the data or NULL if the thread is forcibly awakened.
- 
- Syntax:       void\*  
               **cyg\_mbox\_timed\_get** (  
                   cyg\_handle\_t mbox,  
                   cyg\_tick\_count\_t timeout  
               );
- Context:       Thread
- Parameters:   mbox—handle to the message box.  
               timeout—absolute time to wait for message box.
- Description:   Attempts to retrieve a message from the specified message box when it is available and returns the address of the data. If the timeout time passes, the message is not retrieved.

NULL is returned if the thread is forcibly awakened. The `timeout` parameter is an absolute time measured in clock ticks. This function is only available when the configuration option *Allow Per-Thread Timers* is enabled.

Syntax: `void*`  
**cyg\_mbox\_tryget** (  
    `cyg_handle_t` mbox  
);

Context: Thread

Parameters: mbox—handle to the message box.

Description: Checks to see if a message is available in the specified message box. If a message is present, it is removed from the message box and the address of the data is returned. If a message is not available or forcibly awakened, NULL is returned.

Syntax: `void*`  
**cyg\_mbox\_peek\_item** (  
    `cyg_handle_t` mbox  
);

Context: Thread

Parameters: mbox—handle to the message box.

Description: Checks to see if a message is available in the specified message box. If a message is present, the address of the data is returned without removing the message from the message box. If no message is available, NULL is returned.

Syntax: `cyg_bool_t`  
**cyg\_mbox\_put** (  
    `cyg_handle_t` mbox,  
    `void *item`  
);

Context: Thread

Parameters: mbox—handle to the message box.

item—message to put in the message box.

Description: Attempts to place a message in the specified message box. If the message box is full, configured by *Message Box Queue Size*, the call will block until the message can be successfully sent; in which case, the call will return TRUE. If the message is not sent successfully, FALSE is returned. This function is only available if the configuration option *Message Box Blocking Put Support* is enabled.

Syntax: `cyg_bool_t`  
**cyg\_mbox\_timed\_put** (  
    `cyg_handle_t` mbox,  
    `void *item`,  
    `cyg_tick_count_t` abstime  
);

Context: Thread

- Parameters: `mbox`—handle to the message box.  
`item`—message to put in the message box.  
`abstime`—absolute time to wait when trying to put a message in the message box.
- Description: Attempts to place a message in the specified message box. If the message is sent successfully, `TRUE` is returned. If the message could not be sent immediately, typically because the message box is full, the function will wait until `abstime` before failing and returning `FALSE`. This function is only available if the configuration options *Message Box Blocking Put Support* and *Allow Per-Thread Timers* are enabled.
- Syntax: `cyg_bool_t`  
**`cyg_mbox_tryput`** (  
`cyg_handle_t mbox,`  
`void *item`  
`);`
- Context: Thread
- Parameters: `mbox`—handle to the message box.  
`item`—message to put in the message box.
- Description: Attempts to put a message in the specified message box. If the message is sent successfully, `TRUE` is returned. If the message could not be sent immediately, typically because the message box is full, `FALSE` is returned.
- Syntax: `cyg_count32`  
**`cyg_mbox_peek`** (  
`cyg_handle_t mbox`  
`);`
- Context: Thread
- Parameters: `mbox`—handle to the message box.
- Description: Returns the number of messages in the specified message box.
- Syntax: `cyg_bool_t`  
**`cyg_mbox_waiting_to_get`** (  
`cyg_handle_t mbox`  
`);`
- Context: Thread
- Parameters: `mbox`—handle to the message box.
- Description: Queries to see if other threads are waiting to receive a message in the specified message box. If so, `TRUE` is returned; otherwise, `FALSE` is returned.
- Syntax: `cyg_bool_t`  
**`cyg_mbox_waiting_to_put`** (  
`cyg_handle_t mbox`  
`);`
- Context: Thread
- Parameters: `mbox`—handle to the message box.
- Description: Queries to see if other threads are waiting to send a message in the specified message box. If so, `TRUE` is returned; otherwise, `FALSE` is returned.

Code Listing 6.6 shows an example using a message box to exchange data between two tasks. The thread and message box initializations are left out in this example.

```
1 #include <cyg/kernel/kapi.h>
2
3 cyg_handle_t mbox_handle;
4
5 // Thread A.
6 //
7 void thread_a(cyg_addrword_t index)
8 {
9 // Run this thread forever.
10 while (1)
11 {
12 // Delay for 1000 ticks.
13 cyg_thread_delay(1000);
14
15 // Send a message to Thread B.
16 cyg_mbox_put(mbox_handle, (void *)12);
17 }
18 }
19
20 //
21 // Thread B.
22 //
23 void thread_b(cyg_addrword_t index)
24 {
25 void *message;
26
27 // Run this thread forever.
28 while (1)
29 {
30 // Wait for the message.
31 message = cyg_mbox_get(mbox_handle);
32
33 // Make sure we received the message before attempting
34 // to process it.
35 if (message != NULL)
36 {
37 // Process the message.
38 }
39 }
40 }
```

**Code Listing 6.6** Message box example code.

Code Listing 6.6 shows an example of Thread A sending a message to Thread B using a message box. Thread A places the message—in this case, the number 12—in the message box





- Parameters: `lock`—pointer to spinlock object.  
`locked`—initial state of spinlock, either locked (TRUE) or unlocked (FALSE).
- Description: Initialize a spinlock.
- Syntax: `void`  
**`cyg_spinlock_destroy`**(  
    `cyg_spinlock_t *lock`  
);
- Context: Any
- Parameters: `lock`—pointer to spinlock object.
- Description: Destroys the specified spinlock.
- Syntax: `void`  
**`cyg_spinlock_spin`**(  
    `cyg_spinlock_t *lock`  
);
- Context: Any
- Parameters: `lock`—pointer to spinlock object.
- Description: Attempt to acquire a spinlock. This function is used when it is known that the current code will not be preempted. For example, if interrupts are disabled or the function is called from an interrupt handler.
- Syntax: `void`  
**`cyg_spinlock_clear`**(  
    `cyg_spinlock_t *lock`  
);
- Context: Any
- Parameters: `lock`—pointer to spinlock object.
- Description: Release a spinlock obtained with the `cyg_spinlock_spin` function.
- Syntax: `cyg_bool_t`  
**`cyg_spinlock_try`**(  
    `cyg_spinlock_t *lock`  
);
- Context: Any
- Parameters: `lock`—pointer to spinlock object.
- Description: Nonblocking attempt to acquire a spinlock. On successful acquisition of the spinlock, TRUE is returned; otherwise, FALSE is returned immediately on failure.
- Syntax: `cyg_bool_t`  
**`cyg_spinlock_test`**(  
    `cyg_spinlock_t *lock`  
);
- Context: Any
- Parameters: `lock`—pointer to spinlock object.
- Description: Determine if the spinlock is locked. Returns TRUE if the spinlock is locked; otherwise, FALSE is returned.

Syntax:       void  
              **cyg\_spinlock\_spin\_intsave**(  
              cyg\_spinlock\_t \*lock,  
              cyg\_addrword\_t istate  
              );

Context:       Any

Parameters:   lock—pointer to spinlock object.  
              istate—previous interrupt state.

Description:   Attempt to acquire a spinlock. This function disables interrupts when the spinlock is acquired. The previous interrupt state is returned in the `istate` parameter. This value should be passed when releasing the spinlock with the `cyg_spinlock_clear_intsave` function.

Syntax:       void  
              **cyg\_spinlock\_clear\_intsave**(  
              cyg\_spinlock\_t \*lock,  
              cyg\_addrword\_t istate  
              );

Context:       Any

Parameters:   lock—pointer to spinlock object.  
              istate—previous interrupt state.

Description:   Release a spinlock obtained with the `cyg_spinlock_spin_intsave` function. The previous interrupt state, obtained with the `cyg_spinlock_spin_intsave` function, is passed in the `istate` parameter.

### 6.3 Summary

We began this chapter with a look at threads and how we use them in the eCos system, including the importance of how we set the stack size for a particular thread based on its execution flow. We then examined the different synchronization mechanisms available in the eCos system. Proper use of the various synchronization mechanisms is important for our applications to operate as intended.

# Other eCos Architecture Components

**T**his chapter details the other software components that are part of the core eCos architecture, including Timing components, Assert and Tracing functionality, and the I/O Control System. The timing components provide different mechanisms for periodic events and are comprised of counters, clocks, alarms, and timers. Asserts and traces provide additional debug functionality so you can build robust embedded systems. Finally, the I/O Control System describes the I/O communication scheme and device driver support.

## 7.1 Counters, Clocks, Alarms, and Timers

Most processor architectures provide a clock or timer mechanism, typically a programmable register, which generates a periodic interrupt. This register is programmed with an initial value that determines how often the interrupt occurs. If the processor architecture does not support an onboard timer mechanism, the platform will have an external source for generating the periodic interrupt.

eCos uses the hardware timer mechanism to drive its timing features, which consist of:

- Counters
- Clocks
- Alarms
- Timers

The kernel uses these timing features to provide time-out, delay, and scheduling services for executing threads. Applications can use the timing features for specific timing-related needs as well.

The HAL provides macros to initialize, reset, and read the hardware device used for the kernel timing features. The implementation of the HAL macros and hardware device used is platform specific. Item List 7.1 describes the HAL clock control macros used for the kernel timing functions.

---

**NOTE** Care must be taken if the `HAL_CLOCK_XXX` macros are used while using the eCos kernel. The eCos kernel uses these macro calls for its own timing-related functions.

#### Item List 7.1 HAL Clock Control Macros

Syntax: **HAL\_CLOCK\_INITIALIZE** (  
     `_period_`  
 )

Parameters: `_period_`—initial value to set the timing device to achieve the desired interrupt rate.

Description: Set the timing device to interrupt at the specified period.

Syntax: **HAL\_CLOCK\_RESET** (  
     `_vector_`,  
     `_period_`  
 )

Parameters: `_vector_`—timing device interrupt vector. On most HAL packages, this parameter is not used.

`_period_`—initial value to set the timing device to achieve the desired interrupt rate.

Description: Reset the timing device with the specified period. This is only necessary for devices that require a reset after the interrupt occurs.

Syntax: **HAL\_CLOCK\_READ** (  
     `_pvalue_`  
 )

Parameters: `_pvalue_`—pointer to counter value read from the timing device.

Description: Reads the current value of the timing device counter since the last interrupt. The hardware counter value is returned in the location pointed to by `_pvalue_`. This macro is hardware dependent and the definition here is the case for most hardware platforms.

The HAL architecture-specific configuration components contain a read-only configuration option describing the real-time clock constants. The HAL real-time clock configuration options can be overridden in the kernel package.

The HAL real-time clock configuration option is located under the platform-specific package and is called *Real-Time Clock Constants*. The read-only suboptions are *Real-Time Clock Numerator* (CYGNUM\_HAL\_RTC\_NUMERATOR), *Real-Time Clock Denominator* (CYGNUM\_HAL\_RTC\_DENOMINATOR), and *Real-Time Clock Period* (CYGNUM\_HAL\_RTC\_PERIOD). Dividing the *Real-Time Clock Numerator* by the *Real-Time Clock Denominator* gives the number of nanoseconds per tick. The *Real-Time Clock Period* is the value that is programmed into the processor's hardware timer such that the timer overflows once per kernel tick. This overflow generates a

hardware interrupt. The *Real-Time Clock Period* is the value passed in the `_period_` parameter in the macros shown in Item List 7.1.

The values for these configuration suboptions are calculated based on the clock source used on the specific target platform. When using an eCos-supported target platform, it is usually not necessary to modify these values.

It might be necessary to modify the real-time clock constants when porting to a new hardware platform. These options can be modified, as described later in this section, using the *Override Default Clock Settings* configuration option located under the *eCos Kernel* package. If this is the case, the *Real-Time Clock Period* is modified according to the specifications of the processor and/or hardware platform. The *Real-Time Clock Numerator* and *Real-Time Clock Denominator* are also modified to reflect the new timer resolution. For example, to increase the frequency by a factor of 10, the *Real-Time Clock Period* is changed in some hardware-defined way. Generally, the period is decreased by a factor of 10 and then the *Real-Time Clock Denominator* is increased by a factor of 10.

The kernel uses its tick to determine the timeslicing interval (`CYGNUM_KERNEL_SCHED_TIMESLICE_TICKS`). The default setting uses five clock ticks per timeslice interval. The kernel real-time clock settings can be overridden by enabling the *Override Default Clock Settings* (`CYGPKG_KERNEL_COUNTERS_CLOCK_OVERRIDE`) configuration option, which is located under the *Counters and Clocks* component in the *eCos Kernel* package. Timeslicing is described further in the *Multilevel Queue Scheduler* section of Chapter 5, *The Kernel*.

Since all kernel-level clock-related operations, such as delays and time-outs, use units of ticks rather than seconds, let's look at the steps for a simple conversion.

1. Determine the delay in nanoseconds. In our example, we want a delay of 60 milliseconds, which is the same as 60,000,000 nanoseconds.
2. Next, we need the clock frequency. In this case, we assume a clock running at 100Hz, which corresponds to 1 tick every 10 milliseconds, or 1 tick every 10,000,000 nanoseconds. This corresponds to a numerator of 100 and a denominator of 1,000,000,000.
3. Finally, we can calculate the tick value we need to use in the call using the equation

$$\frac{\text{Delay (in nanoseconds)} \times \text{Numerator}}{\text{Denominator}} = \text{Clock ticks.}$$

Therefore, in our example we plug in the values and get

$$\frac{60000000 \times 100}{1000000000} = 6.$$

4. We then call the clock-related kernel function and pass it the parameter 6 for our 60-millisecond delay.

One more thing to remember, these conversion calculations can sometimes be computation intensive. Therefore, it is usually a good idea in an embedded system to perform these calculations whenever possible at compile time rather than at run time.

The kernel can be configured to provide a *Real-Time Clock* (RTC) for the system. The RTC is necessary to support clock- and alarm-related functions such as `cyg_thread_delay`. It is also needed for the multilevel queue scheduler when using timeslicing. Item List 7.2 details the kernel clock configuration options.

The kernel uses the `HAL_CLOCK_INITIALIZE` macro when it initializes the real-time clock. `HAL_CLOCK_RESET` is used in the ISR for the real-time clock.

### Item List 7.2 Kernel Clock Configuration Options

|             |                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Provide Real-Time Clock</b>                                                                                                                                                                                                                         |
| CDL Name    | <code>CYGVAR_KERNEL_COUNTERS_CLOCK</code>                                                                                                                                                                                                              |
| Description | Allows the kernel to provide the real-time clock for clock- and alarm-related functions and timeslicing (when using the multilevel queue scheduler). The default setting for this option is enabled.                                                   |
| Option Name | <b>Override Default Clock Settings</b>                                                                                                                                                                                                                 |
| CDL Name    | <code>CYGPKG_KERNEL_COUNTERS_CLOCK_OVERRIDE</code>                                                                                                                                                                                                     |
| Description | Allows overriding of the default clock calculations for a particular platform. The default settings attempt to configure 100 clock interrupts per second. The default setting for this option is disabled.                                             |
| Option Name | <b>Measure Real-Time Clock Interrupt Latency</b>                                                                                                                                                                                                       |
| CDL Name    | <code>CYGVAR_KERNEL_COUNTERS_CLOCK_LATENCY</code>                                                                                                                                                                                                      |
| Description | Measures the latency of the real-time clock timer interrupt. This option requires the HAL macro <code>HAL_CLOCK_LATENCY</code> to be defined. The default setting for this option is disabled. This option is only for a performance measurement.      |
| Option Name | <b>Measure Real-Time Clock DSR Latency</b>                                                                                                                                                                                                             |
| CDL Name    | <code>CYGVAR_KERNEL_COUNTERS_CLOCK_DSR_LATENCY</code>                                                                                                                                                                                                  |
| Description | Measures the DSR latency for the real-time clock timer interrupt. This option requires the HAL macro <code>HAL_CLOCK_LATENCY</code> to be defined. The default setting for this option is disabled. This option is only for a performance measurement. |

The kernel contains default settings for the clock interrupt frequency that are specific to each platform. The default RTC frequency is 100Hz; however, you should consult the documentation for the specific platform you are using to verify this value. The RTC settings are derived from the clock source provided on the target hardware. The configuration option *Override Default Clock Settings* contains three configuration suboptions:

- **Clock Hardware Initialization Value** (`CYGNUM_KERNEL_COUNTERS_CLOCK_OVERRIDE_PERIOD`)—initial value programmed into the programmable hardware timer register that generates the periodic interrupts for the kernel timing features.
- **Clock Resolution Numerator** (`CYGNUM_KERNEL_COUNTERS_CLOCK_OVERRIDE_NUMERATOR`)—numerator value for calculating the resolution of clock interrupts in nanoseconds.

- **Clock Resolution Denominator** (CYGNUM\_KERNEL\_COUNTERS\_CLOCK\_OVERRIDE\_DENOMINATOR)—denominator value for calculating the resolution of clock interrupts in nanoseconds.

The resolution is represented as a numerator and denominator value to minimize the drift for frequencies that cannot be expressed as an integer. These suboptions allow you to override the default resolution of the hardware timing device. Overriding this value affects the operation of the real-time clock.

### 7.1.1 Counters

The first eCos timing feature is a *counter*. A counter is an abstraction, which maintains an increasing value that is driven by a source of ticks. The source of the tick does not have to be from a hardware device, nor does the tick need to be periodic. However, it is up to the owner of the counter to ensure that the counter object is being ticked.

eCos offers two different implementations of the counter object. The first implementation uses a single linked list for maintaining alarms attached to counters. When a tick occurs, the kernel goes through this linked list, usually at the DSR level. Therefore, if there is a sizeable number of alarms attached to a single counter object, the system dispatch latency is affected.

The second implementation uses a table of linked lists for maintaining alarms attached to counters, allowing the size of the table to be set by a configuration suboption. This can improve the responsiveness of the kernel because only one list is searched per tick; however, extra code and data is required. The counter configuration options are described in Item List 7.3.

#### Item List 7.3 Counter Configuration Options

|             |                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Implement Counters Using a Single List</b>                                                                                                                                                                                                                                                                                                                                                                             |
| CDL Name    | CYGIMP_KERNEL_COUNTERS_SINGLE_LIST                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | Uses a single linked list for maintaining alarm objects. This is a more efficient use of memory when a small number of alarms are used in the system. The default setting for this option is enabled.                                                                                                                                                                                                                     |
| Option Name | <b>Implement Counters Using a Table of Lists</b>                                                                                                                                                                                                                                                                                                                                                                          |
| CDL Name    | CYGIMP_KERNEL_COUNTERS_MULTI_LIST                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | Uses a table of linked lists for alarm objects. This option reduces the amount of computation necessary when a timer triggers, which is useful when many alarms are used in the system. This option is disabled by default. When using a table of lists the suboption <i>Size of Counter List Table</i> , which has a default value of 8, can be configured. The range for the counter list table size is from 1 to 1024. |
| Option Name | <b>Sort the Counter List</b>                                                                                                                                                                                                                                                                                                                                                                                              |
| CDL Name    | CYGIMP_KERNEL_COUNTERS_SORT_LIST                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | Allows the list of alarms that are attached to counters to be sorted so that the next alarm to trigger is at the front of the list. This reduces the amount of work that needs to be done when                                                                                                                                                                                                                            |



a counter tick is processed. This option causes the operation of adding alarms to the list more expensive because the list must be sorted. The default setting for this option is disabled.

The eCos kernel API provides functions for controlling counters. The counter API functions are detailed in Item List 7.4.

#### Item List 7.4 Kernel Counter API Functions

Syntax:       void  
               **cyg\_counter\_create**(  
                   cyg\_handle\_t \*counter,  
                   cyg\_counter \*the\_counter  
                   );

Context:       Init/Thread

Parameters:   counter—pointer to new counter handle.  
               the\_counter—pointer to the new counter object.

Description:   Construct a new counter.

Syntax:       void  
               **cyg\_counter\_delete**(  
                   cyg\_handle\_t counter  
                   );

Context:       Init/Thread

Parameters:   counter—handle to the counter.

Description:   Remove the specified counter. A counter should never be deleted if a clock or alarm object is attached.

Syntax:       void  
               **cyg\_counter\_tick**(  
                   cyg\_handle\_t counter  
                   );

Context:       Init/Thread/DSR

Parameters:   counter—handle to the counter.

Description:   Increment the counter value by one tick.

Syntax:       cyg\_tick\_count\_t  
               **cyg\_counter\_current\_value**(  
                   cyg\_handle\_t counter  
                   );

Context:       Init/Thread/DSR

Parameters:   counter—handle to the counter.

Description:   Returns the current value, in ticks, of the specified counter.

Syntax:       void  
               **cyg\_counter\_set\_value**(  
                   cyg\_handle\_t counter,  
                   cyg\_tick\_count\_t new\_value

```
);
```

Context:       Init/Thread/DSR

Parameters:    counter—handle to the counter.  
              new\_value—value, in ticks, to set counter.

Description:   Sets the counter to the tick value specified by new\_value.

Code Listing 7.1 shows an example of a counter that causes an alarm to trigger. We discuss alarms later in this chapter.

```
1 #include <cyg/kernel/kapi.h>
2
3
4 cyg_counter counter_obj;
5 cyg_handle_t counter_hdl;
6
7 cyg_handle_t alarm_hdl;
8 cyg_alarm alarm_obj;
9
10 // Declare the alarm handler function so it can
11 // be passed into the alarm initialize function.
12 cyg_alarm_t alarm_handler;
13
14 unsigned long index = 0;
15
16 //
17 // Counter thread.
18 //
19 void counter_thread(cyg_addrword_t index)
20 {
21
22 // Run forever.
23 while (1)
24 {
25 // Delay for 10 ticks.
26 cyg_thread_delay(10);
27
28 // Increment the counter.
29 cyg_counter_tick(counter_hdl);
30 }
31 }
32
33 //
34 // Main starting point for the application.
35 //
36 void cyg_user_start(void)
37 {
38 // Create the counter.
```

```
39 cyg_counter_create(&counter_hdl,
40 &counter_obj);
41
42 // Create the alarm.
43 cyg_alarm_create(counter_hdl,
44 alarm_handler,
45 (cyg_addrword_t)index,
46 &alarm_hdl,
47 &alarm_obj);
48
49 // Initialize the alarm.
50 cyg_alarm_initialize(alarm_hdl,
51 12,
52 6);
53
54 // Create and run the counter thread.
55 }
56
57 //
58 // Alarm handler.
59 //
60 void alarm_handler(
61 cyg_handle_t alarm_handle,
62 cyg_addrword_t data)
63 {
64 (unsigned long)data++;
65 }
```

**Code Listing 7.1** Example code using counters and alarms.

In Code Listing 7.1, a counter is created on line 39. Next, the alarm is created using the previously created counter handle, `counter_hdl`, as shown on line 43. The function that is called when the alarm triggers, `alarm_handler`, is passed in the parameter on line 44. The variable `index`, on line 45, is passed to the `alarm_handler` when the alarm triggers. The alarm handle, `alarm_hdl` on line 46, and alarm object, `alarm_obj` on line 47, are returned after the alarm is created successfully.

Now the alarm can be initialized using the alarm handle we just created, as shown on line 50. Line 51 is the value that the counter, `counter_hdl`, must reach before the alarm first triggers; in this case, the value is 12. On line 52 is the interval that causes the alarm to trigger again. For this alarm initialization, the alarm triggers again when the counter reaches 18, and 24, and 30, and so on—since the interval is set at 6.

The thread creation is eliminated from this code because we covered that in previous examples. After the `counter_thread` is running, it delays for 10 ticks (line 26) and then increments the `counter_hdl` counter (line 29) using the `cyg_counter_tick` function call.

When the `counter_hdl` counter reaches 12 ticks, the `alarm_handler` function is called on line 60. In this case, the `alarm_handler` function simply increments the `data` parameter passed in as the second parameter, which in turn increments the variable `index`.

### 7.1.2 Clocks

A *clock* is a counter, with an associated resolution, which is driven by a regular source of ticks that represent time periods. The eCos kernel implements a default system clock, the RTC, which tracks real time. Item List 7.5 lists the kernel API functions for clock control. Code Listing 7.2 shows an example using the kernel clock API functions along with the kernel alarm API functions.

#### Item List 7.5 Kernel Clock API Functions

Syntax:       void  
              **cyg\_clock\_create**(  
                  cyg\_resolution\_t resolution,  
                  cyg\_handle\_t \*handle,  
                  cyg\_clock \*clock  
              );

Context:       Init/Thread

Parameters:   resolution—numerator and denominator value in nanoseconds per tick.  
              handle—pointer to the new clock handle.  
              clock—pointer to the new clock object.

Description:   Construct a new clock with the specified resolution.

Syntax:       void  
              **cyg\_clock\_delete**(  
                  cyg\_handle\_t clock  
              );

Context:       Init/Thread

Parameters:   clock—handle to the clock.

Description:   Remove the specified clock.

Syntax:       void  
              **cyg\_clock\_to\_counter**(  
                  cyg\_handle\_t clock,  
                  cyg\_handle\_t \*counter  
              );

Context:       Init/Thread/DSR

Parameters:   clock—handle to the clock.  
              counter—pointer to the new counter handle.

Description:   Converts a clock handle to counter handle allowing the use of kernel counter API functions. This gives access to the clock's attached counter.

Syntax:       void  
              **cyg\_clock\_set\_resolution**(  
                  cyg\_handle\_t clock,

```

 cyg_resolution_t resolution
);

```

Context: Init/Thread/DSR

Parameters: `clock`—handle to the clock.  
`resolution`—numerator and denominator value in nanoseconds per tick.

Description: Changes the resolution of the specified clock object. This function does not actually change the behavior of the hardware driving the clock. Instead, `cyg_clock_set_resolution` synchronizes the kernel clock object to match resolution of the underlying hardware clock providing the ticks.

Syntax:

```

 cyg_resolution_t
cyg_clock_get_resolution(
 cyg_handle_t clock
);

```

Context: Init/Thread/DSR

Parameters: `clock`—handle to the clock.

Description: Returns the current resolution of the specified clock.

Syntax:

```

 cyg_handle_t
cyg_real_time_clock(
 void
);

```

Context: Init/Thread/DSR

Parameters: None

Description: Returns a handle to the system real-time clock.

Syntax:

```

 cyg_tick_count_t
cyg_current_time(
 void
);

```

Context: Init/Thread/DSR

Parameters: None

Description: Returns the real-time clock counter value in ticks.

### 7.1.3 Alarms

Another eCos timing feature is the *alarm*. An alarm is attached to a counter and provides a means for generating events based on the value of a counter. The event can be configured to trigger periodically or once.

When an alarm is configured, a handler function is used to perform the necessary processing for handling the event. The alarm handler must follow the same guidelines as other DSR functions, which are detailed in Chapter 3, *Exceptions and Interrupts*, in the section *Interrupt and Scheduler Synchronization*.

Item List 7.6 details the kernel alarm API functions. An example, in the file `simple-alarm.c`, is provided that shows an implementation of an alarm using the real-time clock. The

eCos examples are provided as part of the installation process and discussed further in Chapter 12, *An Example Application Using eCos*.

#### Item List 7.6 Kernel Alarm API Functions

- Syntax:       void  
              **cyg\_alarm\_create**(  
                  cyg\_handle\_t counter,  
                  cyg\_alarm\_t \*alarm\_fn,  
                  cyg\_addrword\_t data,  
                  cyg\_handle\_t \*handle,  
                  cyg\_alarm \*alarm  
                  );
- Context:       Init/Thread  
Parameters:     counter—handle to counter which alarm is attached.  
                  alarm\_fn—pointer to alarm handler function.  
                  data—parameter passed into alarm handler.  
                  handle—pointer to the new alarm handle.  
                  alarm—pointer to the new alarm object.
- Description:   Construct an alarm object that is attached to the specified counter. The alarm handler is called when the alarm triggers and executes in the context of the function that incremented the counter. The alarm does not run until after the call to `cyg_alarm_initialize`.
- Syntax:       void  
              **cyg\_alarm\_delete**(  
                  cyg\_handle\_t alarm  
                  );
- Context:       Init/Thread  
Parameters:     alarm—handle to the alarm.
- Description:   Disables the specified alarm and detaches it from the counter.
- Syntax:       void  
              **cyg\_alarm\_initialize**(  
                  cyg\_handle\_t alarm,  
                  cyg\_tick\_count\_t trigger,  
                  cyg\_tick\_count\_t interval  
                  );
- Context:       Init/Thread/DSR  
Parameters:     alarm—handle to the alarm.  
                  trigger—tick value that causes alarm to occur.  
                  interval—tick value that causes alarm to reoccur. Setting this parameter to zero disables the alarm after it occurs once.
- Description:   Initializes the specified alarm to trigger when the tick value is equal to the `trigger` parameter. If the `interval` parameter is set to zero, the alarm is disabled after it occurs once. Otherwise, the alarm reoccurs according to the `interval` parameter setting.
- Syntax:       void  
              **cyg\_alarm\_enable**(

```

 cyg_handle_t alarm
);

```

Context: Init/Thread/DSR  
Parameters: alarm—handle to the alarm.  
Description: Enables the specified alarm, allowing it to occur in phase with the original settings from the `cyg_alarm_initialize` function.

```

Syntax: void
 cyg_alarm_disable(
 cyg_handle_t alarm
);

```

Context: Init/Thread/DSR  
Parameters: alarm—handle to the alarm.  
Description: Disables the specified alarm preventing it from occurring. The alarm can be re-enabled using the `cyg_alarm_initialize` or `cyg_alarm_enable` functions.

In Code Listing 7.2, we see an example using the kernel clock API along with the kernel alarm API.

```

1 #include <cyg/kernel/kapi.h>
2
3 cyg_handle_t counter_hdl;
4 cyg_handle_t sys_clk;
5 cyg_handle_t alarm_hdl;
6 cyg_tick_count_t ticks;
7 cyg_alarm_t alarm_handler;
8 cyg_alarm alarm_obj;
9
10 unsigned long index;
11
12 //
13 // Main starting point for the application.
14 //
15 void cyg_user_start(void)
16 {
17 sys_clk = cyg_real_time_clock();
18
19 cyg_clock_to_counter(sys_clk,
20 &counter_hdl);
21
22 cyg_alarm_create(counter_hdl,
23 alarm_handler,
24 (cyg_addrword_t)&index,
25 &alarm_hdl,
26 &alarm_obj);
27
28 cyg_alarm_initialize(alarm_hdl,

```

```
29 cyg_current_time() + 100,
30 100);
31 }
32
33 //
34 // Alarm handler.
35 //
36 void alarm_handler(
37 cyg_handle_t alarm_handle,
38 cyg_addrword_t data)
39 {
40 (unsigned long)data++;
41 }
```

**Code Listing 7.2** Example code using clocks and alarms.

Code Listing 7.2 is an example of how to use an alarm with the system real-time clock. In the `cyg_user_start` function, shown on line 15, a handle to the real-time clock is stored in `sys_clk` using the function `cyg_real_time_clock`, as shown on line 17.

Next, we get access to the real-time clock's attached counter, on line 19. The handle is stored in the variable `counter_hdl`, as we see on line 20. On line 22, we use the handle to the system real-time clock to create an alarm. When the alarm triggers, the function passed in on line 23, `alarm_handler`, is called and the function is passed the variable `index`, shown on line 24. The alarm handle is returned in the parameter passed in on line 25, `alarm_hdl`. The alarm object is stored in the parameter passed on line 26, `alarm_obj`.

Finally, we initialize the alarm on line 28 using the alarm handle we just created. In the `cyg_alarm_initialize` function call, we set the initial trigger of the alarm on line 29. In this case, we use the `cyg_current_time` function call, which returns the current real-time clock counter value, and add 100 to the tick value. This causes the alarm to trigger in 100 ticks from the current time. The parameter on line 30 determines the interval to trigger the alarm after the initial trigger. In this case, the `alarm_handler` function is called every 100 real-time clock ticks.

The `alarm_handler` function is shown on lines 36 through 41, which simply increments the `data` parameter passed into the function and in turn increments the `index` variable setup when the alarm was created on line 24.

#### 7.1.4 Timers

A *timer* is an alarm that is attached to a clock. There is a timer object defined by the kernel. However, eCos does not provide a formal implementation, or kernel API functions, for the timer object.

Timers in the eCos system are used within the  $\mu$ ITRON compatibility layer package. The  $\mu$ ITRON package uses the timer object attached to the real-time clock for performing its needed timing related functions.



## 7.2 Asserts and Tracing

eCos supports two mechanisms to aid in debugging—*asserts* and *tracing*. An assert is a piece of code that checks, at run time, whether a condition is expected. If the condition is not expected, an error message can be output and the application is halted. Assertions can determine if there is a bug in the code and isolate the problem immediately, rather than having the application fail later during execution.

Tracing allows the output of text messages at various points in the application's execution. This output enables you to follow the execution flow of a program or check a particular status when certain events occur.

The eCos assert support is complementary to the ISO C standard assert functionality contained in the *Assertions Implementation Header* (`CYGBLD_ISO_ASSERT_HEADER`) configuration option under the *ISO C and POSIX Infrastructure* package.

Both asserts and traces are defined as macros. The first parameter to the macro is a Boolean that determines whether the message is output. Leaving the assertion and tracing code enabled can cause performance degradation in a released image. Using macros allows the code to be compiled in during application debug, however, the associated overhead is easily removed for a released image. The assert macros are defined in the file `cyg_ass.h` within the *infra* package. The trace macros are defined in the file `cyg_trac.h`, also within the *infra* package. This file also contains additional details in the comments at the top of the file about the trace macros and their usage.

Assert and tracing messages are output on the port configured with the *Diagnostic Serial Port* (`CYGNUM_HAL_VIRTUAL_VECTOR_CONSOLE_CHANNEL`) configuration option located in the architecture-specific *eCos HAL* package.

There are three basic assertion macros defined by eCos that can be used within an application when assertions are enabled. Each offers control over the output message content. The first text output by all assert macros is a standard message "ASSERT FAIL", which might be followed by a possible additional text message. The three basic assert macros are:

- **CYG\_FAIL**—does not accept a condition as its first parameter. Instead, this macro outputs the standard message along with a possible user-defined message regardless of any conditions being met.
- **CYG\_ASSERT**—daccepts a condition as its first parameter. Depending on the value of the condition, this macro outputs the standard message along with a possible user-defined message.
- **CYG\_ASSERTC**—dcompact version of the assertion macro that outputs the standard message along with the resulting value of the first parameter.

eCos supports four different modes for assert and trace messages. These modes determine the format of the information that is output. The four modes are:

- Null
- Simple
- Fancy
- Buffered

The buffered tracing mode does not output the message until `CYG_TRACE_PRINT` is called. Item List 7.7 gives a description of the output messages for each type of message output mode.

There is also a mechanism for obtaining the kernel state using the `CYG_TRACE_DUMP` macro, which is also defined in the file `cyg_trac.h`.

eCos defines five different types of trace macros. The various macros provide standard mechanisms for outputting different trace messages. The trace macros have the form `CYG_TRACE###`, where # defines the number of arguments (zero through nine) passed into the macro for outputting with the trace message, and ### defines the format for the argument output. eCos defines trace macros to allow up to eight arguments for output. Table 7.1 shows the different trace macros along with a description.

**Table 7.1** Trace Macros

| Macro Name                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>CYG_TRACE0</code> through <code>CYG_TRACE8</code>                                                                                                                                   | First parameter is a Boolean that determines whether the trace message is output. The other possible arguments are output using printf-style formatting.                                                                                                                                                                                            |
| <code>CYG_TRACE1X</code> through <code>CYG_TRACE8X</code><br><code>CYG_TRACE1Y</code> through <code>CYG_TRACE8Y</code><br><code>CYG_TRACE1D</code> through <code>CYG_TRACE8D</code>       | First parameter is a Boolean that determines whether the trace message is output.<br>X—outputs arguments using <code>%08x</code> format.<br>Y—outputs arguments using <code>%x</code> format.<br>D—outputs arguments using <code>%d</code> format.                                                                                                  |
| <code>CYG_TRACE1XV</code> through <code>CYG_TRACE8XV</code><br><code>CYG_TRACE1YV</code> through <code>CYG_TRACE8YV</code><br><code>CYG_TRACE1DV</code> through <code>CYG_TRACE8DV</code> | First parameter is a Boolean that determines whether the trace message is output. X, Y, and D have the same formats as defined previously. V causes the argument name to be output in the trace message. For example:<br><code>CYG_TRACE1XV(var);</code><br>would output the following trace message:<br><code>TRACE:file.c[8]rout(): var=12</code> |

**Table 7.1** Trace Macros (Continued)

| Macro Name                                                                                                        | Description                                                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CYG_TRACE1XB through CYG_TRACE8XB<br>CYG_TRACE1YB through CYG_TRACE8YB<br>CYG_TRACE1DB through CYG_TRACE8DB       | B means that there is no first parameter Boolean; therefore, using this trace macro always results in a message output. X, Y, and D have the same formats as defined previously.                                                               |
| CYG_TRACE1XVB through CYG_TRACE8XVB<br>CYG_TRACE1YVB through CYG_TRACE8YVB<br>CYG_TRACE1DVB through CYG_TRACE8DVB | B means that there is no first parameter Boolean; therefore, using this trace macro always results in a message output. X, Y, and D have the same formats as defined previously. V causes the argument name to be output in the trace message. |

The trace macros provide a means for tailoring the level of trace messages within an application. This allows control of the amount of output messages during runtime. The trace level can be controlled by the first parameter passed into the trace macro. Code Listing 7.3 shows an example of using the trace-level macros to control the output messages during debugging.

```

1 #include <cyg/infra/cyg_trac.h>
2
3 static int trace_level = 1;
4
5 #define TL1 (0 < trace_level)
6 #define TL2 (1 < trace_level)
7
8 void my_routine(
9 unsigned long index)
10 {
11 unsigned char v1, v2, v3;
12
13 index++;
14
15 // Processing using local variables v1, v2, and v3.
16
17 CYG_TRACE1(TL1, "Index: %d", index);
18
19 CYG_TRACE3(TL2, "Locals: %d %d %d", v1, v2, v3);
20 }

```

**Code Listing 7.3** Trace output runtime control example.

As we see from Code Listing 7.3, the variable `trace_level`, shown on line 3, controls which trace messages are output. The different `TLX` macros, on lines 5 and 6, define the trace levels for the messages.

The function `my_routine` is passed in a parameter named `index`, which is incremented on line 13. The local variables are used in some processing as shown on line 15. The two trace messages on lines 17 and 19 allow different levels of information to be output depending on the trace-level setting.

In this example, the `CYG_TRACE1` message on line 17 is output because the `TL1` macro (on line 5) has a value of 1 when `trace_level` is set to 1. However, the `CYG_TRACE3` message on line 19 is not output because the `TL2` macro (on line 6) has a value of 0 when `trace_level` is set to 1.

Changing the value of `trace_level` to a value of 2, which can be done at run time, allows the `CYG_TRACE3` message to be output the next time `my_routine` is called.

The main configuration option *Asserts & Tracing* (`CYGPKG_INFRA_DEBUG`), located within the *Infrastructure* package, determines whether any assert or trace messages are included in the application image. By default, asserts and tracing are disabled. Item List 7.7 lists the assert and trace configuration options available in the eCos system.

#### Item List 7.7 Assertion and Tracing Configuration Options

Option Name **Use Asserts**

CDL Name `CYGDBG_USE_ASSERTS`

Description Enables assertion code checking and output messages.

Option Name **Use Tracing**

CDL Name `CYGDBG_USE_TRACING`

Description Enables trace code output messages.

Option Name **Null Output**

CDL Name `CYGDBG_INFRA_DEBUG_TRACE_ASSERT_NULL`

Description Disables output messages for tracing and assertion functions. This enables breakpoints to be placed in the trace and assert routines during a debug session instead of interpreting output messages.

Option Name **Simple Output**

CDL Name `CYGDBG_INFRA_DEBUG_TRACE_ASSERT_SIMPLE`

Description Specifies the message format for assert and trace output. This format includes the thread identification number, the filename, line number, routine name, and any additional text message.

Option Name **Fancy Output**

CDL Name `CYGDBG_INFRA_DEBUG_TRACE_ASSERT_FANCY`

Description Specifies the message format for assert and trace output. This format includes the thread identification number, the filename, line number, routine name, and any additional text message.

Option Name **Buffered Tracing**

CDL Name `CYGDBG_INFRA_DEBUG_TRACE_ASSERT_BUFFER`

**Description** Allows tracing and assertion messages to be stored in a buffer. These messages are output when `CYG_TRACE_PRINT` is called. Suboptions define the buffer size and whether the buffer wraps, halts, or outputs when it is full. The trace buffer can also be configured to output when an assertion occurs.

**Option Name** **Use Function Names**

**CDL Name** `CYGDBG_INFRA_DEBUG_FUNCTION_PSEUDOMACRO`

**Description** Allows trace and assert macros to include the function name in output messages. Although this is helpful to read during debug, this option increases the code size.

The *Use Asserts* configuration option defines four suboptions that enable different forms of the assert macro. The first suboption is called *Preconditions* (`CYGDBG_INFRA_DEBUG_PRECONDITIONS`), which is used to determine if a condition is met prior to proceeding.

The second suboption is called *Postconditions* (`CYGDBG_INFRA_DEBUG_POSTCONDITIONS`), which checks that a condition is met at the end of a piece of code, typically before a function returns.

Another suboption is called *Loop Invariants* (`CYGDBG_INFRA_DEBUG_LOOP_INVARIANTS`), which is used to determine if a condition is true for every iteration through a loop.

The final suboption is called *Use Assert Text* (`CYGDBG_INFRA_DEBUG_ASSERT_MESSAGE`). This option allows you to insert your own message within the assert macro output to aid in debugging.

---

**NOTE** It is important to realize that outputting text messages using the tracing functionality in eCos can cause a significant increase in your application code size.

The *Use Tracing* configuration option also defines suboptions. The first suboption is called *Trace Function Reports* (`CYGDBG_INFRA_DEBUG_FUNCTION_REPORTS`). This suboption enables individual trace output messages for entry and exit of functions.

The other suboption is called *Use Trace Text* (`CYGDBG_INFRA_DEBUG_TRACE_MESSAGE`), which, similar to the assert suboption, allows additional text to be embedded into the trace output message.

### 7.3 ISO C and Math Libraries

The eCos *ISO C library* package provides compatibility with the International Organization for Standardization (ISO) 9899:1990 (also known as American National Standards Institute (ANSI) C3.159-1989) specification for the standard C library. This library does not include mathematical functions. Instead, eCos provides a separate *math library* that incorporates these mathematical functions. These libraries allow you to use well-known standard C functions. By default, all ISO C support is thread safe.

The ISO C standard output uses the diagnostic console device provided by the HAL. This is controlled by the *Default Console Device* (`CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE`)

configuration option. The HAL diagnostic device uses a polling mode for communication. This means that output can be slow especially when communicating with a GDB host, which involves utilizing the GDB remote protocol. Input can be processing intensive, meaning that other threads might not have an opportunity to run.

For better performance, an interrupt driven driver should be used. Enabling the proper hardware device driver, such as `"/dev/ser0"`, using the *Hardware Serial Device Drivers* (CYGPKG\_IO\_SERIAL\_DEVICES) configuration package and then layering a TTY-mode driver, such as `"/dev/tty0"`, over the hardware device driver accomplishes this. The *TTY-Mode Serial Device Drivers* (CYGPKG\_IO\_SERIAL\_TTY) configuration package controls the TTY-mode drivers. The serial device drivers are located under the *Serial Device Drivers* (CYGPKG\_IO\_SERIAL) package. The *Default Console Device* configuration option is then set to use `"/dev/tty0"`. The serial and TTY device driver names might vary based on the hardware platform.

It is currently not possible to receive console input when using GDB to debug an application; for example, using `scanf`. Instead, either GDB should not be used or a different HAL diagnostic device needs to be used for communication.

The source code for the ISO C library package is found under the subdirectory `languages\c\libc`. The math library source code is found under the `languages\c\libm` subdirectory.

eCos provides configuration option packages for the ISO C library, which are described in Item List 7.8. These configuration option packages are found under the *ISO C Library* package.

#### Item List 7.8 ISO C Library Configuration Option Packages

|             |                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>ISO C Library Internationalization Functions</b>                                                                                                          |
| CDL Name    | CYGPKG_LIBC_I18N                                                                                                                                             |
| Description | Allows configuration of ISO C internationalization functions such as <code>ctype.h</code> and locale-related functions.                                      |
| Option Name | <b>ISO C Library <code>setjmp/longjmp</code> Functions</b>                                                                                                   |
| CDL Name    | CYGPKG_LIBC_SETJMP                                                                                                                                           |
| Description | Allows configuration of the build options for the <code>setjmp.h</code> functions.                                                                           |
| Option Name | <b>ISO C Library Signal Functions</b>                                                                                                                        |
| CDL Name    | CYGPKG_LIBC_SIGNALS                                                                                                                                          |
| Description | Specifies the configuration of the signal functionality within the ISO C library, such as the <code>signal</code> and <code>raise</code> functions.          |
| Option Name | <b>ISO C Environment Startup/Termination Functions</b>                                                                                                       |
| CDL Name    | CYGPKG_LIBC_STARTUP                                                                                                                                          |
| Description | Controls the configuration of startup, such as the <code>main</code> entry point, and termination, such as <code>exit</code> , for full ISO C compatibility. |

|             |                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------|
| Option Name | <b>ISO C Library Standard Input/Output Functions</b>                                                           |
| CDL Name    | CYGPKG_LIBC_STDIO                                                                                              |
| Description | Allows configuration of the input/output functions found in the <code>stdio.h</code> library file.             |
| Option Name | <b>ISO C Library General Utility Functions</b>                                                                 |
| CDL Name    | CYGPKG_LIBC_STDLIB                                                                                             |
| Description | Specifies the configuration options for the utility functions found in the <code>stdlib.h</code> library file. |
| Option Name | <b>ISO C Library String Functions</b>                                                                          |
| CDL Name    | CYGPKG_LIBC_STRING                                                                                             |
| Description | Controls the configuration options for the string functions found in the <code>string.h</code> library file.   |
| Option Name | <b>ISO C Library Date and Time Functions</b>                                                                   |
| CDL Name    | CYGPKG_LIBC_TIME                                                                                               |
| Description | Configures the ISO C date and time functions.                                                                  |

There are four compatibility modes, which deal with how errors are handled, available for the math library:

- **ANSI/POSIX 1003.1**—the function `matherr` is never called; warning messages are not printed out on the `stderr` output stream; `errno` is set correctly.
- **IEEE-754**—the function `matherr` is never called; warning messages are not printed out on the `stderr` output stream; `errno` is never set.
- **X/Open Portability Guide Issue 3 (XPG3)**—the function `matherr` is called; warning messages are not printed out on the `stderr` output stream; `errno` is set correctly.
- **System V Interface Definition Edition 3**—the function `matherr` is called; warning messages are printed out on the `stderr` output stream; `errno` is set correctly; functions that overflow return a value, which is the maximum single precision floating-point value.

The math library compatibility mode configuration options are found in the *Compatibility Mode* component under the *Math Library* package. The default compatibility mode is POSIX. The compatibility mode can be set at run time. The *Compatibility Mode Setting* configuration option under the *Thread Safety* component allows the setting of the math library compatibility mode, a thread-safe operation. This option is disabled by default.

## 7.4 I/O Control System

The eCos *I/O control system* is comprised of two modules, the *I/O Sub-System* and the *Device Drivers*. The eCos design supports multiple instances of the same type of device present in the system; for example, certain platforms might contain two serial or Ethernet ports. However, platforms

that do not contain multiple devices do not incur any additional overhead in the system. The eCos I/O control system is written entirely in C.

The eCos I/O control system modules are comprised of packages that are configured like other components. These packages can be added or removed to support the specific hardware device needs for the application. The steps for adding and removing packages are covered in Chapter 11, *The eCos Toolset*.

The I/O Sub-System packages are located in the `io` subdirectory, and the device driver packages are in the `devs` subdirectory. Each device type contains a subdirectory under both modules. For example, the Ethernet subsystem package is located in the `io\eth` subdirectory, and the different Ethernet device driver packages are located under the `devs\eth` subdirectory.

The device driver package subdirectories are generally separated by architecture and device; for example, the PowerPC Fast Ethernet Controller device driver package is located under the `devs\eth\powerpc\fec` subdirectory. Figure 1.3, in Chapter 1 shows the structure of the two I/O control system subdirectories. The devices supported are for specific target platform hardware; however, the code can be adapted to other more generic devices. The device driver packages include support for:

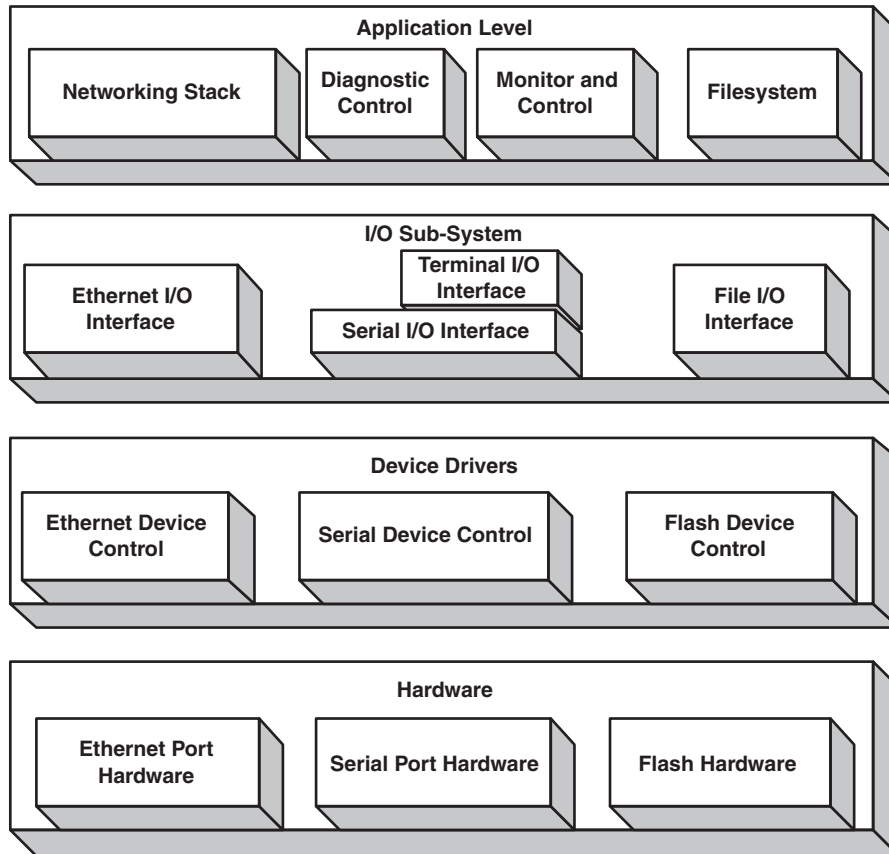
- Ethernet
- Flash
- Compaq IPAQ Platform-Specific Keyboard
- Compaq IPAQ Platform-Specific Touch Screen
- Personal Computer Memory Card International Association (PCMCIA)
- Serial
- Universal Serial Bus (USB)
- Watchdog
- Wallclock

The I/O control system design uses a layered approach. This enables each module to offer basic and device-specific I/O functionality and present it to higher-level software components. Components can be layered on top of each other, in some cases, to extend the functionality of a particular device.

An example of this, shown in Figure 7.1, is the Terminal I/O Interface that makes use of the simple Serial I/O Interface. The Terminal I/O Interface extends the simple serial functionality by providing line buffering and editing. This modularized design also allows individual packages to be configured independently of other device components in the system. In Figure 7.1, we can see an example of an I/O control system configuration. This example is not of any particular platform; it is intended to show the software layers within the I/O control system. In this example, there are three hardware devices in the system—Ethernet, serial, and flash.

Starting at the lowest level are the hardware devices. In some cases, such as a serial port, the device might be contained within the processor itself. Above the hardware are the device drivers. These contain the specific software implementations for controlling their respective





**Figure 7.1** Example eCos I/O control system.

devices. Next is the I/O Sub-System, which presents a generalized interface to the application level for controlling individual hardware devices. At the top is the application level, which contains components such as the networking stack that might use the Ethernet or serial ports, or a file system that might make use of the flash device. The application-level components can be optionally implemented based on the functionality needed in the system. The application-level components present their own programming interface.

#### 7.4.1 I/O Sub-System

The I/O Sub-System provides a standard API for accessing low-level hardware devices. Access to the device drivers is accomplished through functions called *handlers*. Device drivers define specific handlers, within their device I/O table entry, based on the type of hardware device supported. These functions are contained in a device I/O table.

The I/O Sub-System functions use a handle to access the device driver. This handle is retrieved from the device I/O table using the `cyg_io_lookup` function, which takes the device name as a parameter. Device names, such as “`/dev/serial0`” or “`/dev/eth1`”, are set up using configuration options. Once the handle for a particular device driver is retrieved, it can be used with the I/O Sub-System API functions. Item List 7.9 defines the I/O Sub-System API functions. These functions return eCos standard error codes that are defined in the file `codes.h` under the `error` subdirectory.

The I/O Sub-System (CYGPKG\_IO) package only has two configuration options available. The first is *Debug I/O Sub-System* (CYGDBG\_IO\_INIT), which enables diagnostic message output during the initialization of the I/O Sub-System interface packages. This option is disabled by default.

The second configuration option, *Basic Support for File Based I/O* (CYGPKG\_IO\_FILE\_SUPPORT), enables simple file I/O functions to support configurations that include the networking stack. This option is disabled when the *File I/O* (CYGPKG\_IO\_FILEIO) package is incorporated into a configuration. This option is enabled by default. The suboption *Number of Open Files* (CYGPKG\_IO\_NFILE) controls the total number of open files in the system.

#### Item List 7.9 I/O Sub-System API Functions

- Syntax:        `Cyg_ErrNo`  
                 **`cyg_io_lookup`**(  
                     `const char *name,`  
                     `cyg_io_handle_t *handle`  
                     `);`
- Parameters:    `name`—device name, typically has the form “`/dev/serial0`”.  
                 `handle`—returned pointer to handle of the device.
- Description:    Looks up the device specified by the `name` parameter in the device table and returns a pointer to the handle, in the `handle` parameter, for the device. If the device is not found in the table, the error `ENOENT` is returned.
- Syntax:        `Cyg_ErrNo`  
                 **`cyg_io_write`**(  
                     `cyg_io_handle_t handle,`  
                     `const void *buf,`  
                     `cyg_uint32 *len`  
                     `);`
- Parameters:    `handle`—handle to the device.  
                 `buf`—pointer to data buffer.  
                 `len`—pointer to the size of data to send. When the function returns, this parameter contains the actual size of data sent.
- Description:    Send data to the device specified by the `handle` parameter. If `ENOERR` is returned, the write operation completed successfully. The actual number of bytes written is returned in the `len` parameter.

- Syntax: `Cyg_ErrNo`  
**cyg\_io\_read**(  
     `cyg_io_handle_t handle,`  
     `void *buf,`  
     `cyg_uint32 *len`  
 );
- Parameters: `handle`—handle to the device.  
`buf`—pointer to the buffer to store the data.  
`len`—pointer to the size of data to receive. When the function returns, this parameter contains the actual size of data received.
- Description: Receive data from the device specified by the `handle` parameter. If `ENOERR` is returned, the write operation completed successfully. The actual number of bytes read is returned in the `len` parameter.
- Syntax: `Cyg_ErrNo`  
**cyg\_io\_get\_config**(  
     `cyg_io_handle_t handle,`  
     `cyg_uint32 key,`  
     `void *buf,`  
     `cyg_uint32 *len`  
 );
- Parameters: `handle`—handle to the device.  
`key`—type of information to retrieve. The `key` values differ for each driver and are defined in the file `config_keys.h` under the `io` subdirectory.  
`buf`—pointer to buffer where data is placed.  
`len`—pointer to size of data to retrieve. When the function returns, this parameter contains the actual size of data retrieved.
- Description: Retrieve the run-time configuration for the device specified by the `handle` parameter. The type of information retrieved is specified by the `key` parameter.
- Syntax: `Cyg_ErrNo`  
**cyg\_io\_set\_config**(  
     `cyg_io_handle_t handle,`  
     `cyg_uint32 key,`  
     `const void *buf,`  
     `cyg_uint32 *len`  
 );
- Parameters: `handle`—handle to the device.  
`key`—type of information to set. The `key` values differ for each driver and are defined in the file `config_keys.h` under the `io` subdirectory.  
`buf`—pointer to data to configure the device.  
`len`—pointer to size of data to set. When the function returns, this parameter contains the actual size of data retrieved.
- Description: Set the run-time configuration for the device specified by the `handle` parameter. The type of configuration information is specified by the `key` parameter.

Code Listing 7.4 shows an example using the I/O Sub-System API.

```
1 #include <cyg/kernel/kapi.h>
2 #include <cyg/io/io.h>
3 #include <cyg/infra/diag.h>
4
5 //
6 // Main starting point for the application.
7 //
8 void cyg_user_start(void)
9 {
10 cyg_io_handle_t tty_hdl;
11 int err;
12 char output_string[] = "Hello There!!!\n";
13 cyg_uint32 output_len = sizeof(output_string);
14
15 err = cyg_io_lookup("/dev/tty0", &tty_hdl);
16
17 if (err)
18 {
19 diag_printf("ERROR opening device tty0.\n");
20 return;
21 }
22
23 err = cyg_io_write(tty_hdl, output_string, &output_len);
24
25 if (err)
26 {
27 diag_printf("ERROR writing to device tty0.\n");
28 return;
29 }
30 }
```

**Code Listing 7.4** I/O Sub-System API example code.

In Code Listing 7.4 we see an example of writing a string to an I/O device. This example assumes the `/dev/tty0` device (CYGPKG\_IO\_SERIAL\_TTY\_TTY0) was enabled, and configured properly for use with a hardware serial port, under the *Serial Device Drivers* (CYGPKG\_IO\_SERIAL) package. We include the I/O Sub-System API in the header file `io.h` as shown on line 2.

The first step to use an I/O device is to obtain a handle to the specified device using the `cyg_io_lookup` function, as shown on line 15. This function is passed the name of the device—in this case, `/dev/tty0`—which was configured in the `CYGPKG_IO_SERIAL_TTY_TTY0` configuration option. A handle to the device is returned upon successful completion of the function call and stored in the variable `tty_hdl`. Before proceeding, we ensure that a valid device handle was returned in our device lookup by checking the error code returned as we see on line 16. If an error occurred, we print out an error message on the diagnostic port (line 19) and return (line 20).

Next, we use the device handle to write out the message string in the variable `output_string` on line 23. The function `cyg_io_write` outputs the number of bytes passed in the third parameter, `output_len`, using the `tty_hdl` device. Again, we check to ensure that the data was written out successfully by checking the return value from the `cyg_io_write` function, as shown on lines 25 through 29.

### 7.4.2 Device Drivers

A *device driver* is a piece of code that controls a specific hardware component. eCos device driver design focuses on efficiency, eliminating any unnecessary complex layering. It is the job of the device driver to isolate and encapsulate the component-specific implementation. This allows the I/O Sub-System to present a standard interface to higher-level software modules using the device I/O table.

The device I/O table is a structure defined as `cyg_devio_table_t` in the file `devtab.h`. This structure defines write, read, get configuration, and set configuration functions for accessing device drivers. The device I/O table is initialized by the `cyg_io_init` function, which is defined in `iosys.c`. This function is called during the HAL startup along with the other constructors. In turn, each device driver initialization function, defined in the driver's device I/O table entry, is called.

For example, when an application needs to output a text message on a serial port, the application simply calls the I/O Sub-System API write function, which in turn uses the appropriate device driver to manipulate the hardware for transmission of each character on the serial line. The application does not need to be aware of any hardware-specific details, such as the registers to program in order to transmit a character out the serial port. Using separate device driver modules in this manner allows the software to be portable across different hardware platforms because the higher-level application code is not dependent on the specific hardware implementation. This modular approach also eases the understanding and debugging of the software.

A device I/O table entry describes eCos device drivers. This structure, `cyg_devtab_entry_t` located in the file `devtab.h`, defines the device name, the device name layered below (if applicable), a pointer to the device I/O table handler functions, the device initialization function, the device I/O table lookup function, and a placeholder for device specific data.

Along with the standard control routines supplied by the device driver, additional functions are provided that are specific to the type of device supported. The device driver also contains the ISR and DSR functions for the device it manages.

Device drivers use an API for interacting with the kernel and HAL. The function parameters and definitions are the same as the non-driver-specific kernel API functions, which do not contain `_drv_` in the function name. The `cyg_drv_isr_lock` and `cyg_drv_isr_unlock` functions are defined the same as the `cyg_interrupt_disable` and `cyg_interrupt_enable` functions, respectively. The function `cyg_drv_dsr_lock` is defined the same as `cyg_scheduler_lock`, and `cyg_drv_dsr_unlock` is defined the same as `cyg_scheduler_unlock`.

For device driver API syntax and function definitions, refer to the kernel API function tables in the previous chapters. The difference between using the kernel API and the driver API is that the driver API is guaranteed to be present in configurations where the eCos kernel is not present. This makes the drivers more portable. The device driver API definitions are located in the file `drv_api.c` and `drv_api.h` under the `hal` subdirectory. Item List 7.10 details the list of device driver API functions.

**Item List 7.10** Device Driver API Functions

```
cyg_drv_isr_lock
cyg_drv_isr_unlock
cyg_drv_dsr_lock
cyg_drv_dsr_unlock
cyg_drv_spinlock_init
cyg_drv_spinlock_destroy
cyg_drv_spinlock_spin
cyg_drv_spinlock_clear
cyg_drv_spinlock_try
cyg_drv_spinlock_test
cyg_drv_spinlock_spin_intsave
cyg_drv_spinlock_clear_intsave
cyg_drv_mutex_init
cyg_drv_mutex_destroy
cyg_drv_mutex_lock
cyg_drv_mutex_trylock
cyg_drv_mutex_unlock
cyg_drv_mutex_release
cyg_drv_cond_init
cyg_drv_cond_destroy
cyg_drv_cond_wait
cyg_drv_cond_signal
cyg_drv_cond_broadcast
cyg_drv_interrupt_create
cyg_drv_interrupt_delete
cyg_drv_interrupt_attach
cyg_drv_interrupt_detach
cyg_drv_interrupt_mask
cyg_drv_interrupt_unmask
cyg_drv_interrupt_acknowledge
cyg_drv_interrupt_configure
cyg_drv_interrupt_level
cyg_drv_interrupt_set_cpu
cyg_drv_interrupt_get_cpu
```

## 7.5 Summary

In this chapter, we began by looking into the different timing features provided in the eCos system. This gave us an understanding of how counters, clocks, and timers are used in our application. We also explored the assert and tracing functionality and how we can use these features during the application debug cycle. We got a basic understanding of the libraries (C and math) included with eCos. Finally, we looked at the I/O Control System and how to use it with the existing device drivers provided with eCos.

## **Additional Functionality and Third-Party Contributions**

**T**he open-source nature of eCos caters to a rich set of extended functionality. This functionality is often provided by external third-party contributors to enhance their own and the open-source community's embedded systems. Included in this functionality and contributions are:

- POSIX, EL/IX, and  $\mu$ ITRON Compatibility Layers
- ROM Monitors
- RAM and ROM File Systems
- PCI Support
- TCP/IP Networking Support
- Embedded Simple Object Access Protocol (SOAP) Toolkit
- Kaffe Java Virtual Machine
- Bluetooth and Wireless Application Protocol (WAP) Support
- Embedded Web Server Support

The eCos Web site maintains a list of the different contributions available and can be found online at:

<http://sources.redhat.com/ecos/contrib.html>

Another source to find the latest information about the latest contributions and functionality available for eCos is the NEWS file. The eCos NEWS file is located in the online source code repository, under the `packages` directory. The online source code repository can be viewed in HTML format at:



<http://sources.redhat.com/cgi-bin/cvsweb.cgi/ecos/?cvsroot=ecos>

In this chapter, we take a look at some of the software components and contributions available for use with eCos. Combining certain components, such as the file system, networking stack, and Web server, you can achieve the desired feature set for many different embedded systems.

## 8.1 Compatibility Layers

Compatibility layers are specifications that define standard APIs that interface to the underlying eCos kernel in order to encapsulate implementation-specific functionality. This allows companies to adopt the API, easing the porting process of applications across different platforms and operating systems. This section gives a brief overview of the compatibility layer support offered with eCos. Resources for additional detailed information are included as well. eCos supports two different compatibility layers:

- POSIX
- $\mu$ ITRON

The packages for these different layers are found under the `compat` subdirectory. This is further divided into the `posix` and `uitron` subdirectories. Figure 1.3, in Chapter 1, shows the overall directory structure.

### 8.1.1 POSIX

The Portable Operating System Interface (POSIX) is a set of Institute of Electrical and Electronics Engineers, Inc. (IEEE) standards designed to ease application portability. The *POSIX* specifications define APIs that detail how applications interface to operating systems.

eCos contains support for the POSIX 1003.1—1996 Specification (ISO/IEC 9945-1). Support for a function means that the data types and definitions necessary to support the particular function, including the objects it manipulates, are also defined. The eCos POSIX support is a subset of the entire POSIX standards, including the implementation of threads, signals, and synchronization objects. Additional information about POSIX can be found in the standard *Portable Operating System Interface (POSIX)—Part 1: System Application Programming Interface (API)[C Language]* ISO/IEC 9945-1:1996, IEEE.

eCos divides the POSIX support into two packages called the *POSIX Compatibility Layer* (CYGPKG\_POSIX) and *POSIX File IO Compatibility Layer* (CYGPKG\_IO\_FILEIO). The POSIX Compatibility Layer package provides support for threads, signals, synchronization, timers, and message queues, while the POSIX File IO Compatibility Layer package provides support for file and device I/O.

eCos defines a template called `Posix` that is used to enable the standard configuration options for POSIX compatibility. Chapter 11, *The eCos Toolset*, describes the process for using templates in greater detail. There are configuration options for both the POSIX Compatibility

Layer and the POSIX File IO Compatibility Layer. The POSIX Compatibility Layer package is made up of the following configuration components:

- POSIX Scheduling Configuration (CYGPKG\_POSIX\_SCHED)
- POSIX Pthread Configuration (CYGPKG\_POSIX\_PTHREAD)
- POSIX Timers (CYGPKG\_POSIX\_TIMERS)
- POSIX Semaphores (CYGPKG\_POSIX\_SEMAPHORES)
- POSIX Message Queues (CYGPKG\_POSIX\_MQUEUES)
- POSIX Signals Configuration (CYGPKG\_POSIX\_SIGNALS)
- POSIX Utsname Configuration (CYGPKG\_POSIX\_UTSNAME)

#### 8.1.1.1 EL/IX

*EL/IX* is an API developed by Red Hat to provide compatibility across different operating systems including Linux, embedded Linux, and eCos. The *EL/IX* API encapsulates subsets of the ISO/IEC 9899:1990 (ISO C) and ISO/IEC 9945-1 (POSIX 1003.1) APIs, as well as a number of other well-known functions commonly found on UNIX and particularly Linux that are suitable for embedded systems.

The *EL/IX* interface ensures application portability for operating systems adopting the standard, which preserves the investment in software development and developer knowledge. Red Hat has initiated the *EL/IX* API as an open-source project. Other companies involved with *EL/IX* include Intel, MIPS, Toshiba, and Pacific Softworks. Additional detailed information about *EL/IX* can be found at:

<http://sources.redhat.com/elix>

Since the base of *EL/IX* functionality is the POSIX specification, operating systems that conform to the POSIX standard should be largely compatible with the *EL/IX* specification.

Functions are present at the associated level and all higher levels. The eCos *EL/IX* package follows Level 1, the RTOS compatibility layer. These various API subsets are different levels defined within *EL/IX*:

- **Level 1**—RTOS Compatible Layer. These functions are available in both Linux and embedded operating systems, such as eCos, RTEMS, VxWorks, and PSOS. Certain functions at this level might have reduced or modified semantics.
- **Level 2**—Linux Single Process Only. Includes Level 1 along with functions from Linux that are not easily implemented on an RTOS. Includes the full implementation of any reduced Level 1 functions.
- **Level 3**—Linux Multiprocess for Embedded Applications. Based on POSIX.1 with the removal of functions not intended for embedded applications.
- **Level 4**—Full POSIX or Linux Compliance. These functions are present in a standard Linux kernel.

### 8.1.2 $\mu$ ITRON

Another compatibility layer eCos supports is called  $\mu$ ITRON. The  $\mu$ ITRON specification defines APIs that enable highly flexible operating system architectures tailored specifically for embedded system applications. One advantage of the  $\mu$ ITRON compatibility layer, as with other compatibility layers, is that the effort of understanding and porting application software to new processor architectures is reduced. A great reference on  $\mu$ ITRON is  *$\mu$ ITRON 3.0, An Open and Portable Real-Time Operating System for Embedded Systems* by Ken Sakamura.

There are four levels of the  $\mu$ ITRON specification:

- **Required (R)**—Functions in this level are mandatory for  $\mu$ ITRON 3.0 implementations.
- **Standard (S)**—Includes basic functions for achieving a real-time, multitasking operating system.
- **Extended (E)**—Includes additional and extended functions, such as object creation and deletion, memory pools, and timer handler functions.
- **CPU Dependent (C)**—Incorporates CPU or hardware configuration implementation-dependent functions.

The eCos  $\mu$ ITRON package supports version 3.02 of the specification, which incorporates all of the Required (R) level functions, all of the Standard (S) level functions, and most of the Extended (E) level functions as well. More detailed information about the  $\mu$ ITRON specification can be found online at:

[www.itron.gr.jp](http://www.itron.gr.jp)

eCos defines a template called *Uitron* that is used to load the packages for  $\mu$ ITRON compatibility. The configuration options for  $\mu$ ITRON are located within the  *$\mu$ ITRON Compatibility Layer* (CYGPKG\_UITRON) package. Included under this package are configuration options for semaphores, mailboxes, tasks, alarm handlers, and others.

## 8.2 ROM Monitors

A *ROM monitor* is a program, typically residing in ROM or flash memory, which provides debug functionality. The ROM monitor is used to load an application program into memory for debugging. After loading the application image, the ROM monitor provides some basic level of debug functionality, such as reading and writing memory or processor registers. The application does not need to provide any debug facilities because this is incorporated into the ROM monitor program.

The eCos system offers several choices for debugging applications. Some of the debugging support options include:

- Use an In-Circuit Emulator (ICE) or other hardware debugging module supported by GDB.
- Include support for GDB directly into the application.
- Use CygMon or RedBoot ROM monitors, which include GDB support, as the resident ROM monitor on the target platform.
- Create a simple application that only includes GDB debugging support known as a GDB stub ROM. This application is programmed into ROM and provides loading and debugging of applications.
- Use a third-party ROM monitor. This ROM monitor must support GDB debugging; otherwise, the application must provide GDB debugging support directly.

The debugging option used is dependent on your preferences and the system resources available.

### 8.2.1 CygMon

*CygMon*, short for Cygnus ROM Monitor, is an eCos supplied standalone ROM monitor program. CygMon is a command-line driven program that provides basic debug functionality, such as program loading and memory content inspection and manipulation. In addition, GDB communication support is included through the use of the GDB stub. CygMon is designed to be portable across all of the supported eCos architectures. Currently, this support includes the `arm`, `mips`, and `mn10300` architectures. CygMon also provides an API for applications, which includes system calls that allow access to serial ports or on-board timers. eCos provides a template, `Cygmon`, for building the ROM monitor program. The CygMon package (`CYGPKG_CYGMON`) is located under the `cygmon` subdirectory.

---

**NOTE** CygMon is no longer maintained in the eCos project. The preferred ROM monitor program to use is RedBoot. Most of the functionality provided by CygMon is contained in the RedBoot ROM monitor. The CygMon monitor can still be used; however, upkeep, such as bug fixes and enhancements, on the code base is being maintained.

### 8.2.2 RedBoot

*RedBoot* is an acronym for Red Hat Embedded Debug and *Bootstrap*. The RedBoot ROM monitor provides a complete bootstrap environment and features such as a flash file system, as well as network downloading and debugging. RedBoot provides its own GDB stub for communication with a GDB host. RedBoot is intended to replace the CygMon ROM monitor and GDB stub ROM debug software. Chapter 9, *The RedBoot ROM Monitor*, provides complete details about using RedBoot.

### 8.2.3 GDB Stub

At the core of the two ROM monitors is the GDB stub. The GDB stub is a piece of software that provides the low-level interaction with the HAL, as well as the GDB protocol communication layer. The low-level HAL interaction includes hooking into the serial port for communication with the GDB host and installing trap handlers for breakpoint support.

The GDB stub can also be built into a simple GDB stub ROM application that resides in the target hardware's ROM and provides the ability to load and debug applications. This application can be used in lieu of a ROM monitor.

The GDB stub code is contained in the file `generic-stub.c` with support routines in the file `hal_stub.c`. These files are located under the `common HAL` subdirectory. For additional information on the GDB communication protocol, see the online documentation at:

[http://sources.redhat.com/gdb/onlinedocs/gdb\\_toc.html](http://sources.redhat.com/gdb/onlinedocs/gdb_toc.html)

GDB stub software can be included in any eCos image to provide GDB debugging functionality. Both CygMon and RedBoot include GDB stubs. The stub code can then be invoked by the GDB host or by the application itself by calling the `breakpoint` routine. There are configuration options that control whether GDB stub code is included. These options are located under the common HAL configuration component *Source-Level Debugging Support*. The configuration options for including GDB stubs in eCos images are detailed in Item List 8.1.

#### Item List 8.1 GDB Stubs Configuration Options

|             |                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Include GDB Stubs in HAL</b>                                                                                                                                                  |
| CDL Name    | <code>CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS</code>                                                                                                                                  |
| Description | Causes the GDB stub code to be included in the image, enabling GDB communication and debug support. The default for this option is disabled.                                     |
| Option Name | <b>Include GDB External Break Support For Stubs</b>                                                                                                                              |
| CDL Name    | <code>CYGDBG_HAL_DEBUG_GDB_BREAK_SUPPORT</code>                                                                                                                                  |
| Description | Enables the GDB stub to include a serial port interrupt handler to listening for GDB break packets. This option allows a target to be stopped asynchronously using the GDB host. |
| Option Name | <b>Include GDB External Break Support When No Stubs</b>                                                                                                                          |
| CDL Name    | <code>CYGDBG_HAL_DEBUG_GDB_CTRL_C_SUPPORT</code>                                                                                                                                 |
| Description | Allows a GDB host to stop a target asynchronously by listening for GDB break packets in a serial port interrupt handler. This option is used when GDB stubs are not present.     |
| Option Name | <b>Include GDB Multi-Threading Debug Support</b>                                                                                                                                 |
| CDL Name    | <code>CYGDBG_HAL_DEBUG_GDB_THREAD_SUPPORT</code>                                                                                                                                 |
| Description | Enables additional HAL code to support multithreaded debugging of an application. This option is enabled by default.                                                             |

Let's look at the two configuration options available that enable the building of a GDB stub ROM image. This image can then be programmed into a target hardware platform for

communication with a GDB host. The configuration options for building a GDB stub image are under the *Global Build Options* component. The stubs package template controls the GDB stubs options.

---

**NOTE** It is best to use a template with the stubs packages when building a GDB stub application rather than individually configuring the GDB stub options. This ensures that the proper packages are loaded and the configuration options are set properly.

The first option is *Build GDB Stub ROM Image* (`CYGBLD_BUILD_GDB_STUBS`), which is disabled by default. When this option is enabled, the platform-specific stub code is built into the GDB stub image. The platform-specific stub code, contained in the files `plf_stub.c` and `plf_stub.h`, provides the necessary functionality to initialize the hardware for GDB stub communication with the hardware. This functionality includes initializing the stub serial port, configuring any Light Emitting Diodes (LED), and setting up the necessary interrupt support for asynchronous program breaking. These routines, if supplied, are called from the common HAL stub function `initHardware`. The GDB stub files produced from this option are `gdb_module.img`, a GDB recognizable file format; `gdb_module.bin`, a binary image; and `gdb_module.srec`, an S-record file. Enabling the platform-specific GDB stub build option requires that the *Build Common GDB Stub ROM Image* option be enabled as well.

The second option, *Build Common GDB Stub ROM Image* (`CYGBLD_BUILD_COMMON_GDB_STUBS`), is disabled by default. Enabling this option causes the common stub file, `stubrom.c`, to be compiled into the image. This file supplies a `cyg_start` routine, which calls the HAL breakpoint function to enter into a GDB debug mode.

### 8.3 File Systems

eCos provides three different file system implementations: ROM, RAM, and JFFS2. File Allocation Table (FAT) support is also being developed.

The Journalling Flash File System version 2 (JFFS2) is a log-structured file system intended for embedded systems containing flash memory devices.

Both the ROM and RAM file system use the *POSIX File I/O Compatibility Layer* (`CYGPKG_IO_FILEIO`) package. The POSIX File I/O package provides control over the file systems installed and can be found in the `fileio` subdirectory under the I/O Sub-System. The POSIX File I/O package contains a file system table array containing entries from each installed file system. The entries in this array are of the type `cyg_fstab_entry`, defined in the file `fileio.h`. Each entry contains information about the file system, such as the name that might have a value of “romfs” for a ROM file system. Also included with each entry are pointers to functions that control directories and files; for example, `make_directory`, `romfs_mkdir`, and `change_directory`, `romfs_chdir`. The maximum number of installed file systems is controlled by a configuration option, as shown in Item List 8.2.

Another table the POSIX File I/O package uses is the mount table, defined in the file `fileio.h` as `cyg_mtab_entry`. This table keeps track of the file systems that are mounted or active. Tables can be mounted statically, using the `MTAB_ENTRY` macro, or dynamically during run time by calling the `mount` function. The maximum number of mounted, statically and dynamically, file systems is controlled by a configuration option; see Item List 8.2 for more details.

When a file is opened, information about the file is stored in the `CYG_FILE_TAG` structure, also defined in the file `fileio.h`. This structure, commonly referred to as a `cyg_file`, keeps track of the state of the file and also contains a pointer to a table of file I/O operations. The file operations include read, write, and seek.

Item List 8.2 describes the configuration options available for the POSIX File I/O package. These options give you control over file system components.

#### Item List 8.2 POSIX Compatibility Layer File I/O Configuration Options

|             |                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Enable Socket Support</b>                                                                                                                                                                                                                           |
| CDL Name    | <code>CYGPKG_IO_FILEIO_SOCKET_SUPPORT</code>                                                                                                                                                                                                           |
| Description | Allows file support for socket interfaces. This option is only valid when the networking package is installed. This option is enabled by default.                                                                                                      |
| Option Name | <b>Maximum Number of Open Files</b>                                                                                                                                                                                                                    |
| CDL Name    | <code>CYGNUM_FILEIO_NFILE</code>                                                                                                                                                                                                                       |
| Description | Controls the maximum number of open files allowed for all file systems. Valid values for this option are 1 to 9,999,999. The default value for this option is 16.                                                                                      |
| Option Name | <b>Maximum Number of Open File Descriptors</b>                                                                                                                                                                                                         |
| CDL Name    | <code>CYGNUM_FILEIO_NFD</code>                                                                                                                                                                                                                         |
| Description | Controls the number of open file descriptors allowed for all file systems. The minimum value for this option is set to the Maximum Number of Open Files; the maximum value is 9,999,999. The default value for this option is 16.                      |
| Option Name | <b>Maximum Number of Installed File Systems</b>                                                                                                                                                                                                        |
| CDL Name    | <code>CYGNUM_FILEIO_FSTAB_MAX</code>                                                                                                                                                                                                                   |
| Description | Sets the maximum number of file systems that the POSIX File I/O package can handle. Valid values for this option are 1 to 9,999,999. The default value for this option is 4.                                                                           |
| Option Name | <b>Maximum Number of Mounted File Systems</b>                                                                                                                                                                                                          |
| CDL Name    | <code>CYGNUM_FILEIO_MTAB_MAX</code>                                                                                                                                                                                                                    |
| Description | Controls the number of mounted file systems handled by the POSIX File I/O package. The minimum value for this option is set to the Number of Dynamically Mounted File Systems; the maximum value is 9,999,999. The default value for this option is 8. |
| Option Name | <b>Number of Dynamically Mounted File Systems</b>                                                                                                                                                                                                      |
| CDL Name    | <code>CYGNUM_FILEIO_MTAB_EXTRA</code>                                                                                                                                                                                                                  |
| Description | Sets the maximum number of mounted file systems that can be created dynamically. Valid values for this option are 0 to 9,999,999. The default value for this option is 4.                                                                              |

Option Name **Maximum Number of Installed Network Stacks**  
 CDL Name `CYGNUM_FILEIO_NSTAB_MAX`  
 Description Controls the maximum number of networking stacks that can be handled by the POSIX File I/O package. Valid values for this option are 1 to 9,999,999. The default value for this option is 1.

Option Name **Enable Current Directory Tracking**  
 CDL Name `CYGPKG_IO_FILEIO_TRACK_CWD`  
 Description Allows the POSIX File I/O package to track the name of the current directory as a string. This supports the `getcwd` function. The default for this option is enabled.

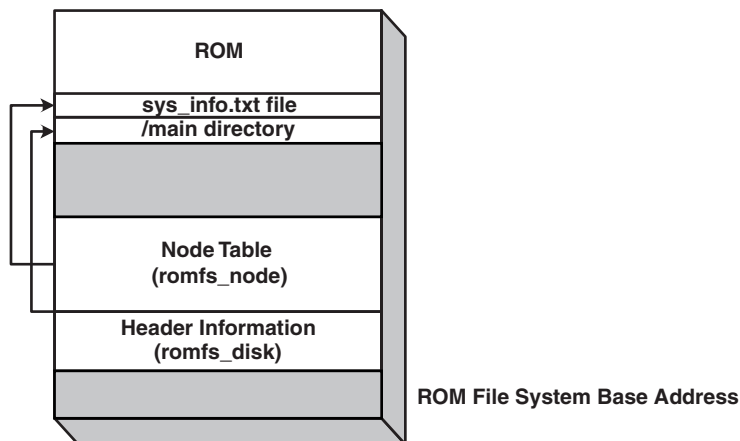
### 8.3.1 ROM File System

The eCos *ROM File System* (`CYGPKG_FS_ROM`) package is located in the `rom` subdirectory under the file system packages subdirectory `fs`. Figure 1.3 in Chapter 1 shows the overall directory structure.

A ROM file system is built on the host development system. This file system is read-only and is stored in the target memory exactly as it was constructed on the host.

At the start of the ROM file system is a header, defined in the file `romfs.c` as `romfs_disk`. This header includes the name of the file system, the size of the file system in bytes, and the total number of nodes in the file system.

Each file and directory in the file system is a node. Nodes are defined by the structure `romfs_node` within the file `romfs.c`. The node structure includes a description of whether the node is a file or directory, the size of data in the node, and the creation time of the file. The data for each file is stored in a contiguous block of memory, which is referenced by the `data` member of the node structure. A table of node objects present in the file system follows the header in memory. Figure 8.1 shows an example ROM file system architecture. We can see the header information and node table location at the base address configuration option. The directory `main` and the file `sys_info.txt` are represented as `romfs_node` objects in the node table.



**Figure 8.1** Example ROM file system memory architecture.



The ROM file system package contains a utility to make a ROM file system image. This code is located in the file `mk_romfs.c` under the `support` subdirectory. The output from this utility is a file `romfs.img`, which contains the files and directories and can be loaded into memory. The ROM file system image utility is built on a host system, such as Windows or Linux. The eCos development tools are not used to build the ROM file system image utility. Additional details about using the ROM file system utility can be found in the file `mk_romfs.txt` in the `doc` subdirectory of the ROM file system package.

### 8.3.2 RAM File System

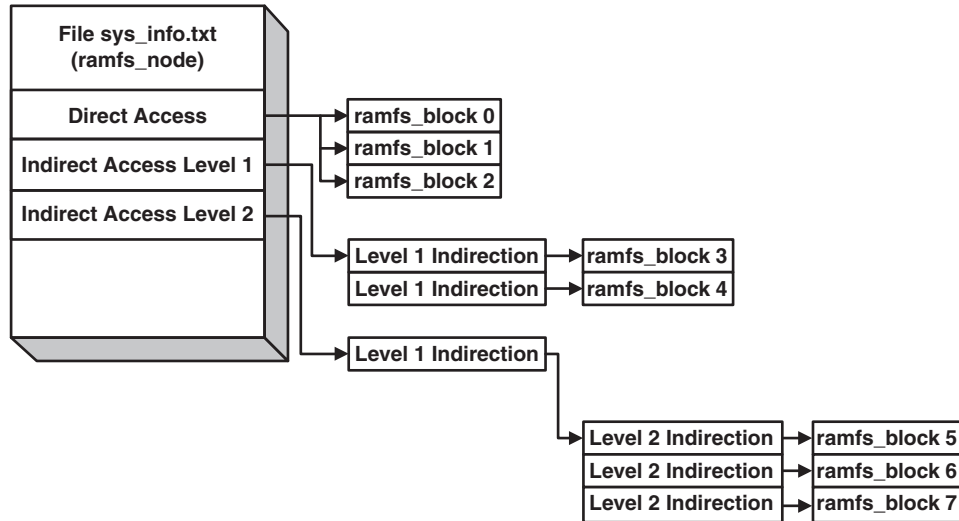
The eCos RAM file system purely uses RAM to store file data. Therefore, it does not permanently store the file system data because the contents are lost when the system is reset. The RAM file system starts off uninitialized with no files present. It can then be used to store temporary data using the classic file system abstraction, for applications requiring a file system, or it can be loaded with the contents of a ROM file system to provide a fully writeable file system.

The eCos *RAM File System* (`CYGPKG_FS_RAM`) package is located in the `ram` subdirectory under the file system packages subdirectory `fs`. Figure 1.3 in Chapter 1 shows the overall directory structure. The RAM file system, similar to the ROM implementation, uses a node to describe files and directories in the system.

The node structure, `ramfs_node`, is defined in the file `ramfs.c` and includes the node type, size of the file in bytes, the last file access time, and last file modification time. The RAM file system implementation contains two different storage mechanisms, Simple and Block.

The *Simple* mechanism uses the `malloc` and `free` routines to allocate memory for nodes and the file data. If the file data increases in size, memory is reallocated for file. This mechanism has the advantage of using only the amount of memory it needs to contain all files in the RAM file system. The Simple mechanism, as the name implies, is a straightforward approach. One disadvantage of this mechanism is fragmentation of the heap from constant reallocations for files. This can lead to the inability of a file to grow because the file size might be larger than the amount of memory that can be allocated from the heap. Another disadvantage is that the `malloc` implementation must be included in the eCos image.

The *Block* mechanism divides file data storage memory into fixed-size blocks. The allocation method for these blocks can be from a predefined array of blocks or using `malloc` and `free`. File data storage for each node, `ramfs_node`, is arranged in up to three arrays of pointers to the data block structure, `ramfs_block`. The number of arrays used is set by configuration options, shown in Item List 8.3. The three arrays allow direct access, single-level indirect access, and two-level indirect access to the data blocks. Figure 8.2 shows the three different data access object arrays for an example file node with a filename `sys_info.txt`.



**Figure 8.2** RAM file system Block data storage mechanism architecture.

The Block data storage mechanism has the advantage of using fixed-size blocks making management of memory easier. Another advantage is that the allocation mechanism is configurable; therefore, the `malloc` implementation can be excluded from the image. A disadvantage of this mechanism is when using the `malloc` allocation method; each block causes a `malloc` function call rather than a single call for the entire file. The memory allocated is also only available for the RAM file system.

There are two configuration options that determine the data storage implementation used for the RAM file system. These options are *Simple Implementation* (`CYGPKG_FS_RAM_SIMPLE`) and *Block-Based Allocation* (`CYGPKG_FS_RAM_BLOCKS`), which are located under the RAM File System package. Only one of the data storage implementations can be selected for the RAM file system. The Simple Implementation contains the configuration suboption that sets the amount of memory to allocate for new data added to a file. This suboption is *Size of File Data Storage Increment* (`CYGNUM_RAMFS_REALLOC_INCREMENT`). The different configuration options for the Block storage mechanism are detailed in Item List 8.3.

### Item List 8.3 RAM File System Block Mechanism Configuration Options

Option Name **Size of File Data Storage Block**  
 CDL Name `CYGNUM_RAMFS_BLOCK_SIZE`  
 Description Controls the size of the `ramfs_block` for file data storage. The valid values for this option are 64 to 32,768 bytes. The default value is 256.

Option Name **Directly Referenced Data Storage Blocks**  
 CDL Name `CYGNUM_RAMFS_BLOCKS_DIRECT`

|             |                                                                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | Sets the size of the array for the directly accessed data storage blocks. The valid values for this option are 0 to 32. The default value is 8.                                                                                                                                                             |
| Option Name | <b>Single Level Indirect Data Storage Blocks</b>                                                                                                                                                                                                                                                            |
| CDL Name    | CYGNUM_RAMFS_BLOCKS_INDIRECT1                                                                                                                                                                                                                                                                               |
| Description | Sets the size of the array for the single-level indirect data storage blocks. The valid values for this option are 0 to 32. Setting this value to 0 eliminates the single-level indirect array from the file node. The default value is 1.                                                                  |
| Option Name | <b>Two Level Indirect Data Storage Blocks</b>                                                                                                                                                                                                                                                               |
| CDL Name    | CYGNUM_RAMFS_BLOCKS_INDIRECT2                                                                                                                                                                                                                                                                               |
| Description | Sets the size of the array for the two-level indirect data storage blocks. The valid values for this option are 0 to 32. Setting this value to 0 eliminates the single-level indirect array from the file node. The default value is 1.                                                                     |
| Option Name | <b>Use Block Array Rather Than Malloc</b>                                                                                                                                                                                                                                                                   |
| CDL Name    | CYGPKG_FS_RAM_BLOCKS_ARRAY                                                                                                                                                                                                                                                                                  |
| Description | Determines whether <code>malloc</code> is used to allocate data storage blocks or an external array block. Using an external array block enables configuration suboptions for setting the size and name of the array. The default for this option is disabled, which causes <code>malloc</code> to be used. |

### 8.3.3 Journalling Flash File System Version 2

eCos also provides support for the Journalling Flash File System, version 2 (JFFS2). JFFS2 is based on JFFS (version 1). JFFS was designed to use embedded flash memory devices more efficiently. JFFS and JFFS2 take into account the characteristics of flash technology when dealing with the typical situation in an embedded system where the system is not always cleanly shut down.

JFFS2 is a log-structured file system, whereas a typical embedded file system emulates a traditional file system that uses block-based storage and keeps track of the files in these blocks. JFFS2 builds on the version 1 technology.

The JFFS2 package (`CYGPKG_FS_JFFS2`) is contained in the `jffs2` directory under the file system's `packages` directory `fs`. The license for the JFFS2 file system can be found in the file `License` under the `src` directory. Additional details about JFFS2 can be found online at:

<http://sources.redhat.com/jffs2>

## 8.4 PCI Support

eCos provides a Peripheral Component Interconnect (PCI) bus library. The PCI bus is a high-performance 32- or 64-bit bus that has multiplexed address and data lines. The bus is intended to connect peripheral controller components, add-in boards, and processor/memory systems. The PCI bus is commonly found in PCs today, but it has also made its way into embedded system designs. The current version of the PCI bus specification is 2.2. Additional information about the PCI specification can be found online at:

[www.pcisig.com](http://www.pcisig.com)

The eCos *PCI Configuration Library* (CYGPKG\_IO\_PCI) package can be found in the `pci` subdirectory under the I/O Sub-System packages subdirectory, `io`. See Figure 1.3 in Chapter 1 for an overall view of the directory structure. This library provides the following functionality:

- Scan the bus for specific devices based on Device and Vendor ID or on a particular device class code.
- Read and modify the generic PCI information.
- Read and modify the device-specific PCI information.
- Allocate PCI memory and I/O space for devices.
- Translate device specific PCI interrupts into HAL vectors.

### 8.4.1 PCI Library API

The eCos PCI library package contains two main source files, `pci.c`, high-level PCI library API functions; and `pci_hw.c`, low-level HAL interface routines. The high-level API routines are used by applications to control the devices on the PCI bus. There are also low-level PCI routines, which are used by the high-level API to access the HAL platform-specific PCI functionality. The high-level PCI library API functions are described in Item List 8.4 and can be found in the file `pci.h`.

#### Item List 8.4 PCI Library API Functions

|              |                                                                                                                                                                                                                                                                                                                                                                                |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax:      | <pre>void <b>cyg_pci_init</b>(     void );</pre>                                                                                                                                                                                                                                                                                                                               |
| Parameters:  | None                                                                                                                                                                                                                                                                                                                                                                           |
| Description: | Initialize the PCI bus allowing access to the configuration space. This should be the first function called, although certain HALs might call this function as part of the platform initialization procedure.                                                                                                                                                                  |
| Syntax:      | <pre>cyg_bool <b>cyg_pci_find_device</b>(     cyg_uint16 vendor,     cyg_uint16 device,     cyg_pci_device_id *devid );</pre>                                                                                                                                                                                                                                                  |
| Parameters:  | <p><code>vendor</code>—Vendor ID of the device to find.</p> <p><code>device</code>—Device ID of the device to find.</p> <p><code>devid</code>—pointer to the Device ID where to start the scan. When the function returns, this points to the Device ID of the next device.</p>                                                                                                |
| Description: | Scans the PCI bus configuration space for a device with the given Vendor and Device IDs. The search begins with the device specified in the <code>devid</code> parameter; specifying <code>CYG_PCI_NULL_DEVID</code> starts the search at the first slot. This function returns <code>TRUE</code> if the specified device is found; otherwise, <code>FALSE</code> is returned. |

- Syntax: `cyg_bool`  
**cyg\_pci\_find\_class**(  
     `cyg_uint32 dev_class,`  
     `cyg_pci_device_id *devid`  
 );
- Parameters: `dev_class`—Class Code of the device to find.  
`devid`—pointer to the bus number, device number, and functional number of the device where to start the scan. When the function returns, this points to the bus number, device number, and functional number of the next device.
- Description: Searches the PCI bus configuration space for the device with the Class Code specified in the `dev_class` parameter. The search begins with the device specified in the `devid` parameter; specifying `CYG_PCI_NULL_DEVID` starts the search at the first slot. This function returns `TRUE` if the specified device is found; otherwise, `FALSE` is returned.
- Syntax: `cyg_bool`  
**cyg\_pci\_find\_next**(  
     `cyg_pci_device_id cur_devid,`  
     `cyg_pci_device_id *next_devid`  
 );
- Parameters: `cur_devid`—bus number, device number, and functional number of the device where the search begins.  
`next_devid`—pointer to the bus number, device number, and functional number of the next device. This parameter can also point to `cur_devid`.
- Description: Scans the PCI configuration space for the next valid device after the device specified in the `cur_devid` parameter. The search begins with the device specified in the `devid` parameter; specifying `CYG_PCI_NULL_DEVID` starts the search at the first slot. This function returns `TRUE` if another device is found; otherwise, `FALSE` is returned.
- Syntax: `cyg_bool`  
**cyg\_pci\_find\_matching**(  
     `cyg_pci_match_func *matchp,`  
     `void *match_callback_data,`  
     `cyg_pci_device_id *devid`  
 );
- Parameters: `matchp`—pointer to function, supplied by the caller, that checks if the device returned matches.  
`match_callback_data`—pointer to user data to pass to the `matchp` callback function.  
`devid`—pointer to device information to begin search.
- Description: Searches the PCI bus configuration space for a device whose properties match those required by the `matchp` function. The `matchp` function is called for each device on the bus. The search begins with the device pointed to in the `devid` parameter. This function returns `TRUE` if a matching device is found; otherwise, `FALSE` is returned.
- Syntax: `void`  
**cyg\_pci\_get\_device\_info**(  
     `cyg_pci_device_id devid,`

```

 cyg_pci_device *dev_info
);

```

Parameters: `dev_id`—bus number, device number, and functional number of the device.

`dev_info`—pointer to returned configuration information for the device.

Description: Gets the generic PCI configuration information for the device specified in the `dev_id` parameter.

Syntax: void  
**cyg\_pci\_set\_device\_info**(  
 cyg\_pci\_device\_id `dev_id`,  
 cyg\_pci\_device\_id \*`dev_info`  
);

Parameters: `dev_id`—bus number, device number, and functional number of the device.

`dev_info`—pointer to configuration information to set the device. The function sets this parameter to the configuration information by reading back the written information when it returns.

Description: Set the generic PCI configuration information for the device specified in the `dev_id` parameter.

Syntax: void  
**cyg\_pci\_read\_config\_uintX**(  
 cyg\_pci\_device\_id `dev_id`,  
 cyg\_uint8 `offset`,  
 cyg\_uintX \*`val`  
);

Parameters: `dev_id`—bus number, device number, and functional number of the device.

`offset`—register offset.

`val`—pointer to the returned register value.

Description: Reads the device-specific register from the PCI configuration space for the device specified in the `dev_id` parameter. The *X* in the function name and the type of the `val` parameter are 8, 16, or 32 and determine the size of the read performed. This routine should be used to access device-specific registers; general accesses should use the `cyg_pci_get_device_info` function.

Syntax: void  
**cyg\_pci\_write\_config\_uintX**(  
 cyg\_pci\_device\_id `dev_id`,  
 cyg\_uint8 `offset`,  
 cyg\_uintX `val`  
);

Parameters: `dev_id`—bus number, device number, and functional number of the device.

`offset`—register offset.

`val`—value to set in the specified register.

Description: Sets the device specific register in the PCI configuration space for the device specified in the `dev_id` parameter. The *X* in the function name and the type of the `val` parameter are 8, 16, or 32 and determine the size of the write performed. This routine should be used to

access device-specific registers; general accesses should use the `cyg_pci_set_device_info` function.

- Syntax: `cyg_bool`  
**`cyg_pci_configure_device`**(  
     `cyg_pci_device *dev_info`  
 );
- Parameters: `dev_info`—pointer to the device configuration header information. When the function returns, this parameter contains the resource allocations for the device.
- Description: Handles all I/O and memory regions that need configuration on a device by allocating memory to all Base Address Registers (BARs). These allocated base addresses are stored in the `dev_info` parameter. If the `dev_info` parameter does not contain valid base size values, `false` is returned. This function also calls `cyg_pci_translate_interrupt`.
- Syntax: `cyg_bool`  
**`cyg_pci_configure_bus`**(  
     `cyg_uint8 bus,`  
     `cyg_uint8 *next_bus`  
 );
- Parameters: `bus`—current bus number.  
`bus_next`—pointer to the subordinate buses. This parameter specifies the bus number to assign to the next subordinate bus found. The number is incremented for new buses discovered.
- Description: Allocates memory and I/O space for all Base Address Registers (BAR) on all devices for the bus specified by the `bus` parameter and subordinate buses specified by the `bus_next` parameter. This function is used in systems with multiple buses connected by bridges. On success, `TRUE` is returned; otherwise, `FALSE` is returned.
- Syntax: `cyg_bool`  
**`cyg_pci_translate_interrupt`**(  
     `cyg_pci_device *dev_info,`  
     `CYG_ADDRWORD *vec`  
 );
- Parameters: `dev_info`—pointer to the device configuration header information.  
`vec`—pointer to translated interrupt vector number.
- Description: Translates the PCI interrupt signal (`INTA\`, `INTB\`, `INTC\` or `INTD\`) to the associated HAL interrupt vector. If the device generates interrupts, the translated vector number is placed in the `vec` parameter and `TRUE` is returned; otherwise, `FALSE` is returned.
- Syntax: `cyg_bool`  
**`cyg_pci_allocate_memory`**(  
     `cyg_pci_device *dev_info,`  
     `cyg_uint32 bar,`  
     `CYG_PCI_ADDRESS64 *base`  
 );

- Parameters: `dev_info`—pointer to the device configuration header information.  
`bar`—Base Address Register to allocate memory.  
`base`—pointer to the base address to allocate the memory. The address of the next free location is returned in this parameter if the allocation succeeds.
- Description: Allocates memory to the BAR specified in the `bar` parameter, which allows a device driver to set up its own memory mappings. If the BAR is the wrong type or the `dev_info` parameter does not contain valid base sizes, `FALSE` is returned.
- Syntax:

```

cyg_bool
cyg_pci_allocate_io(
 cyg_pci_device *dev_info,
 cyg_uint32 bar,
 CYG_PCI_ADDRESS32 *base
);

```
- Parameters: `dev_info`—pointer to the device configuration header information.  
`bar`—Base Address Register to allocate I/O space.  
`base`—pointer to the base address to allocate the I/O space. The address of the next free location is returned in this parameter if the allocation succeeds.
- Description: Allocates I/O space to the BAR specified in the `bar` parameter, which allows a device driver to set up its own I/O mappings. If the BAR is the wrong type or the `dev_info` parameter does not contain valid base sizes, `FALSE` is returned.
- Syntax:

```

void
cyg_pci_set_memory_base(
 CYG_PCI_ADDRESS64 base
);

```
- Parameters: `base`—address for BAR memory mapping.
- Description: Set the base address for memory mapping, overriding the default values set in the platform PCI initialization.
- Syntax:

```

void
cyg_pci_set_io_base(
 CYG_PCI_ADDRESS32 base
);

```
- Parameters: `base`—address for BAR I/O mapping.
- Description: Set the base address for I/O mapping, overriding the default values set in the platform PCI initialization.

A good source for example code on using the PCI library API functions can be found in the PCI Library package test file `pci1.c`. Within this file is the routine `pci_test`. This routine shows how to initialize the PCI bus and then scan the bus for all devices present.

## 8.5 USB Support

Universal Serial Bus (USB) networks consist of a single host, called the Host Controller, and one or more slave devices. Slave devices connect to the USB through ports on specialized USB



devices called hubs. The host initiates all USB operations. For example, if a host wants to receive data from a USB peripheral, the host issues an IN token to the peripheral. The slave peripheral then responds with the data or a NAK. USB slave peripherals cannot interact with each other. The current version of the USB specification is 2.0. USB supports four different types of communication:

- **Control Transfers**—consist of standard, class, vendor, and reserved. All devices must respond to certain standard control messages.
- **Interrupt Transfers**—limited-latency transfer to or from a device. The data can be presented for transfer at any time and is delivered by the USB at a rate that is not slower than what is specified by the device. An example of this type of transfer is the coordinates from a USB mouse device.
- **Isochronous Transfers**—intended for multimedia devices with large amounts of data that is continuously exchanged.
- **Bulk Transfers**—typically consists of larger amounts of data. Reliability of this data is ensured by the hardware level using error detection and retries. Bandwidth for bulk transfers varies depending on other bus activities.

Additional detailed information about the USB and the necessary specifications can be found online at:

[www.usb.org](http://www.usb.org)

eCos provides support for USB slave devices. The eCos USB support consists of four different packages, which only provide support for developing USB slave peripherals and not USB hosts. USB slave device driver packages are located under the `usb` subdirectory. This is with the other device drivers within the `devs` subdirectory. The other packages for USB support are located under the `usb` subdirectory within the I/O Sub-System subdirectory `io`. Figure 1.3 in Chapter 1 gives an overall view of the directory structure. The USB packages are:

- **USB Device Drivers**—contains the specific implementations for the USB slave hardware devices supported. Currently, the USB device driver support exists for the Intel StrongARM SA-11x0, and has the package name *SA11X0 USB Device Driver* (`CYGPKG_DEVS_USB_SA11X0`).
- **Common USB**—provides information common to the host and slave sides of the bus such as the details of the control protocol. The *USB Support* (`CYGPKG_IO_USB`) package is located under the `common` subdirectory within the I/O Sub-System.
- **Common USB Slave**—defines the USB API for device drivers and provides utilities, such as control message handlers, needed by the device drivers and applications. The file `usbs.h` contains the USB API. The *USB Slave-Side Support*

(`CYGPKG_IO_USB_SLAVE`) package is located under the `slave` subdirectory within the I/O Sub-System.

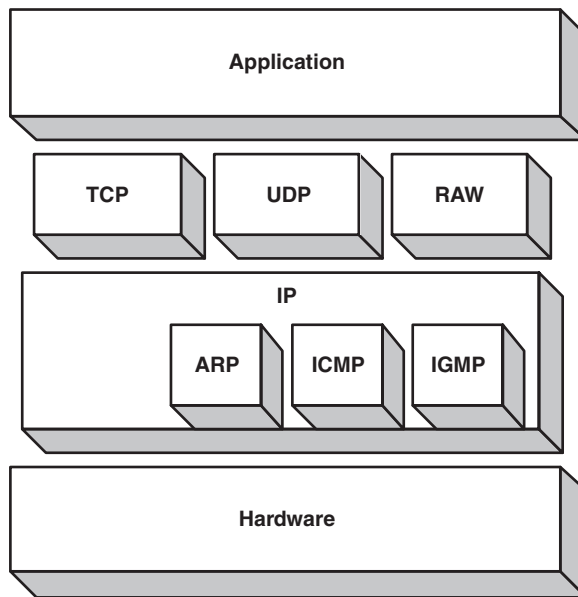
- **Class-Specific USB Support**—eases the development of specific classes of USB peripherals. Currently, USB-Ethernet devices are supported. The package *USB Slave Ethernet Support* (`CYGPKG_IO_USB_SLAVE_ETH`) is located under the `eth` subdirectory within the I/O Sub-System.

Additional information about the USB support in eCos can be found online at:

<http://sources.redhat.com/ecos/docs.html>

## 8.6 Networking Support

Internet connectivity for embedded devices is quickly becoming a standard requirement. From Internet Appliances to remote sensor controllers, the ability to send and receive data over the Internet, or a private network, is becoming standard. The core to this connectivity is the network stack. Figure 8.3 shows a general architecture diagram of the some of the protocols supported by the eCos Networking package.



**Figure 8.3** eCos general networking architecture.

Networking stacks use several different protocols for communication. For example, the Simple Mail Transfer Protocol (SMTP) is used to send and receive email. The eCos Networking package incorporates the frequently used standard protocols for network communication. Using these protocols, a wide range of application layer components can be developed, such as a mail

transfer module for remote email notifications or a Web server module for remote status and control, which meet the needs of many embedded devices. Additional information about the networking stack, including the networking stacks API, can be found online at:

<http://sources.redhat.com/ecos/docs.html>

Several sources are available that detail the different protocols commonly used for network communications. One good source is the book *TCP/IP Illustrated, Volume 1: The Protocols* by W. Richard Stevens.

Another source for detailed information about Internet protocols is the Request For Comment (RFC) repository. RFCs are documents that set standards for the Internet community. Several sites are available for browsing through these documents; one RFC site is:

[www.ietf.org/rfc.html](http://www.ietf.org/rfc.html)

There is a Basic Networking Framework package provided with eCos for networking support. This Basic Networking Framework package is modular, which allows the selection of the actual stack implementation as a configuration option. There are two different network stack implementations available. One is adapted from OpenBSD and the other from FreeBSD.

### 8.6.1 OpenBSD

The first implementation is derived from the OpenBSD networking code. OpenBSD is a UNIX-like open-source operating system project based on 4.4BSD source code. Various components are incorporated in the OpenBSD project, including a networking stack. Additional information about OpenBSD can be found online at:

[www.openbsd.org](http://www.openbsd.org)

The protocols supported in the OpenBSD implementation are:

- **IPv4**—Internet Protocol version 4
- **ARP**—Address Resolution Protocol
- **RARP**—Reverse Address Resolution Protocol
- **ICMP**—Internet Control Message Protocol
- **UDP**—User Datagram Protocol
- **TCP**—Transmission Control Protocol
- **DHCP**—Dynamic Host Configuration Protocol
- **BOOTP**—Bootstrap Protocol
- **TFTP**—Trivial File Transfer Protocol

In addition, the OpenBSD implementation also supports raw packet interface. There are other features supported, but untested, by the OpenBSD network stack. These other features include:

- IPv6—Internet Protocol version 6
- Berkeley Packet Filter (BPF)
- Generic Tunneling Interface (GIF)
- Multicast support, which includes Multicast routing

The source code for the OpenBSD network stack implementation is located in the `tcpip` subdirectory under the `net` root directory.

The eCos Net template includes the OpenBSD implementation. Using the Net template allows you to incorporate the OpenBSD network stack into your application image, which includes the necessary device drivers for a particular platform. See Chapter 11 for more information about using templates. The OpenBSD networking package (`CYGPKG_NET_OPENBSD_STACK`) can also be added independently to customize the network configuration for a specific image build.

The license terms of the OpenBSD stack can be found online at:

[www.openbsd.org/policy.html](http://www.openbsd.org/policy.html)

### 8.6.2 FreeBSD

The other networking implementation is derived from FreeBSD networking code released from the KAME project. FreeBSD is based on the 4.4BSD-Lite operating system. Additional information about FreeBSD can be found online at:

[www.freebsd.org](http://www.freebsd.org)

Additional information about the KAME project can be found online at:

[www.kame.net](http://www.kame.net)

The networking protocols supported in the FreeBSD implementation include:

- **IPv4**—Internet Protocol version 4
- **IPv6**—Internet Protocol version 6
- **ARP**—Address Resolution Protocol
- **RARP**—Reverse Address Resolution Protocol
- **ICMP**—Internet Control Message Protocol
- **IGMP**—Internet Group Management Protocol
- **UDP**—User Datagram Protocol
- **TCP**—Transmission Control Protocol
- **DHCP**—Dynamic Host Configuration Protocol
- **BOOTP**—Bootstrap Protocol
- **TFTP**—Trivial File Transfer Protocol
- **Multicast addressing**

Along with these protocols, the FreeBSD implementation also supports raw packet interface. FreeBSD also supports other features in its network stack; however, these features are untested. These other features include:

- Berkeley Packet Filter (BPF)
- Multicast routing

The source code for the FreeBSD network stack implementation is located in the `bsd_tcpip` subdirectory under the `net` root directory.

eCos also has a template available to include the FreeBSD implementation called `New_Net`. However, the FreeBSD networking package (`CYGPKG_NET_FREEBSD_STACK`) can be included as an individual component as well. Chapter 11 contains details about using templates and packages in eCos.

The FreeBSD license terms can be found online at:

[www.freebsd.org/copyright/copyright.html](http://www.freebsd.org/copyright/copyright.html)

### 8.6.3 lwIP

There is also a network stack contribution available called Lightweight TCP/IP (lwIP), which is basically a small memory footprint implementation of the TCP/IP protocol suite. The focus of the lwIP implementation is to provide full TCP/IP support while reducing the necessary memory footprint. lwIP supports IP, ICMP, UDP, and TCP and is available under a BSD-style license. Additional information about the lwIP contribution can be found online at:

[www.sics.se/~adam/lwip](http://www.sics.se/~adam/lwip)

### 8.6.4 Networking Threads

The eCos networking code uses threads to accomplish particular tasks. It is important to understand what networking threads are running in the system to determine the resource needs when using the networking stack.

It is also necessary to keep the thread priorities for the networking code in mind when designing the overall thread priority scheme to ensure that adequate network performance is achieved. Certain threads can be enabled or disabled via the Basic Networking Framework package configuration options or under the specific stack implementation, OpenBSD or FreeBSD, configuration options. Item List 8.5 describes the networking threads, their default priority levels, and the configuration options used to set the specific priority level.

#### Item List 8.5 Networking Threads

|                        |                                              |
|------------------------|----------------------------------------------|
| Thread Function        | <b>alarm_thread</b>                          |
| Filename               | <code>timeout.c</code>                       |
| CDL Name               | <code>CYGPKG_NET_FAST_THREAD_PRIORITY</code> |
| Default Priority Level | 6                                            |

|                        |                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description            | Services transmit and receive interrupts for Ethernet network device driver. This thread also handles network timeout events. Setting this thread's priority level higher than other application thread priority levels allows the best network performance. Setting the priority level lower can affect the throughput of network traffic. This thread is enabled by default. |
| Thread Function        | <b>cyg_netint</b>                                                                                                                                                                                                                                                                                                                                                              |
| Filename               | support.c                                                                                                                                                                                                                                                                                                                                                                      |
| CDL Name               | CYGPKG_NET_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                                     |
| Default Priority Level | 7                                                                                                                                                                                                                                                                                                                                                                              |
| Description            | Handles ARP requests and IP (version 4) incoming datagram parsing. Also processes IPv6 incoming datagram parsing and Ethernet bridge frames when these options are enabled. This background thread priority level affects the network stack performance. This thread is enabled by default.                                                                                    |
| Thread Function        | <b>cyg_rs</b>                                                                                                                                                                                                                                                                                                                                                                  |
| Filename               | ipv6_routing_thread.c                                                                                                                                                                                                                                                                                                                                                          |
| CDL Name               | CYGINT_NET_IPV6_ROUTING_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                        |
| Default Priority Level | 8                                                                                                                                                                                                                                                                                                                                                                              |
| Description            | Handles sending router solicitation messages used to exchange IP addresses and other information. This thread is disabled by default since IPv6 is also disabled by default.                                                                                                                                                                                                   |
| Thread Function        | <b>dhcp_mgt_entry</b>                                                                                                                                                                                                                                                                                                                                                          |
| Filename               | dhcp_support.c                                                                                                                                                                                                                                                                                                                                                                 |
| CDL Name               | CYGPKG_NET_DHCP_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                                |
| Default Priority Level | 8                                                                                                                                                                                                                                                                                                                                                                              |
| Description            | Manages the DHCP leases for each network interface by ensuring that each interface is configured and functioning properly. If a problem occurs or a lease expires, the DHCP thread reinitializes and reconfigures the network interfaces. This thread is enabled by default with DHCP support.                                                                                 |
| Thread Function        | <b>tftpd_server</b>                                                                                                                                                                                                                                                                                                                                                            |
| Filename               | tftp_server.c                                                                                                                                                                                                                                                                                                                                                                  |
| CDL Name               | CYGPKG_NET_TFTPD_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                               |
| Default Priority Level | 10                                                                                                                                                                                                                                                                                                                                                                             |
| Description            | A TFTP server that manages client applications attempting to connect and perform read and write requests. The TFTP is used to exchange files among devices connected to the Internet. Using the TFTP server requires the inclusion of a file system implementation. This thread is enabled by default.                                                                         |

### 8.6.5 Networking Configuration

The eCos *Basic Networking Framework* package (CYGPKG\_NET) is located in the `common` subdirectory under the `net` subdirectory. See Figure 1.3 in Chapter 1 for an overall view of the repository directory structure. The Basic Networking Framework package configuration options are described in Item List 8.6.

**Item List 8.6 Basic Networking Framework Package Configuration Options**

|             |                                                                                                                                                                                                                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>INET Support</b>                                                                                                                                                                                                                                                                                                     |
| CDL Name    | CYGPKG_NET_INET                                                                                                                                                                                                                                                                                                         |
| Description | Allows the use of IPv4 and ARP within the networking stack. This option is enabled by default. This option contains suboptions for enabling IPv6 Support (CYGPKG_NET_INET6), Multicast Routing Support (CYGSEM_NET_ROUTING), and the Use Random Sequence (CYGSEM_NET_RANDOMID). All suboptions are disabled by default. |
| Option Name | <b>TFTP (RFC-1350) Support</b>                                                                                                                                                                                                                                                                                          |
| CDL Name    | CYGPKG_NET_TFTP                                                                                                                                                                                                                                                                                                         |
| Description | Includes client and server library support for TFTP. This option is enabled by default. The Priority Level for TFTP Daemon Thread (CYGPKG_NET_TFTPD_THREAD_PRIORITY) suboption sets the priority level of the TFTP server thread. The default priority level is 10.                                                     |
| Option Name | <b>Use Full DHCP Instead of BOOTP</b>                                                                                                                                                                                                                                                                                   |
| CDL Name    | CYGPKG_NET_DHCP                                                                                                                                                                                                                                                                                                         |
| Description | Enables the use of DHCP for the initialization of the IP address for network interfaces. Otherwise, BOOTP is used, which does not require as much resources as DHCP. This option is enabled by default. When this option is enabled, suboptions configure the DHCP management thread settings.                          |
| Option Name | <b>Options Controlling IPv6 Routing</b>                                                                                                                                                                                                                                                                                 |
| CDL Name    | CYGPKG_NET_IPV6_ROUTING                                                                                                                                                                                                                                                                                                 |
| Description | The configuration suboptions control the IPv6 thread for sending router solicitation messages, such as thread priority and the rate at which the solicitations are sent out. This option is only enabled by default when the IPv6 Support configuration suboption is enabled.                                           |
| Option Name | <b>Debug Output</b>                                                                                                                                                                                                                                                                                                     |
| CDL Name    | CYGPKG_NET_DEBUG                                                                                                                                                                                                                                                                                                        |
| Description | Enables diagnostic output for various stack operations. The default for this option is disabled.                                                                                                                                                                                                                        |
| Option Name | <b>Network Timing Statistics</b>                                                                                                                                                                                                                                                                                        |
| CDL Name    | CYGPKG_NET_TIMING_STATS                                                                                                                                                                                                                                                                                                 |
| Description | Controls the diagnostic output of timing messages related to various stack function calls such as memcpy and mbuf_alloc. The default for this option is disabled.                                                                                                                                                       |
| Option Name | <b>Build Networking Tests (Demo Programs)</b>                                                                                                                                                                                                                                                                           |
| CDL Name    | CYGPKG_NET_BUILD_TESTS                                                                                                                                                                                                                                                                                                  |
| Description | Enables network tests to be built. This option is disabled by default.                                                                                                                                                                                                                                                  |
| Option Name | <b>Initialization Options for 'eth0'</b>                                                                                                                                                                                                                                                                                |
| CDL Name    | CYGHWR_NET_DRIVER_ETH0_SETUP_OPTIONS                                                                                                                                                                                                                                                                                    |
| Description | This component contains configuration options that specify the method (DHCP/BOOTP or static) for initializing the Ethernet 0 interface. The default is to use DHCP/BOOTP for initialization.                                                                                                                            |

Option Name **Initialization Options for 'eth1'**  
CDL Name CYGHWDR\_NET\_DRIVER\_ETH1\_SETUP\_OPTIONS  
Description This component contains configuration options that specify the method (DHCP/BOOTP or static) for initializing the Ethernet 1 interface, if it exists. The default is to use DHCP/BOOTP for initialization.

The two networking stack implementations (OpenBSD and FreeBSD) also contain their own configuration options. The configuration options with XXX in the CDL name correspond to either FREEBSD for the FreeBSD stack implementation or OPENBSD for the OpenBSD stack implementation. These configuration options are detailed in Item List 8.7.

#### **Item List 8.7** Network Stack Implementation Specific Package Configuration Options

Option Name **Implement the Socket API Locally**  
CDL Name CYGPKG\_NET\_API\_LOCAL  
Description Defines whether the networking API functions, such as `bind`, `socket`, and `connect`, are supplied by the stack itself package.

Option Name **Implement the Socket API Via FILEIO Package**  
CDL Name CYGPKG\_NET\_API\_FILEIO  
Description Defines whether the networking API functions, such as `bind`, `socket`, and `connect`, are supplied by the FILEIO package.

Option Name **INET Support**  
CDL Name CYGPKG\_NET\_XXX\_INET  
Description Enables IPv4. This option is enabled by default. The suboption for enabling IPv6 Support (CYGPKG\_NET\_XXX\_INET6) is disabled by default.

Option Name **Number of BPF Filters**  
CDL Name CYGPKG\_NET\_NBPF  
Description Sets the number of BPFs. The BPF code, which allows sniffing of the network packets, is not included in the eCos release, but can be obtained from the OpenBSD site. The default for this option is 0.

Option Name **Built-in Ethernet Bridge Code**  
CDL Name CYGPKG\_NET\_BRIDGE  
Description Enables the code for Ethernet bridge support. The default for this option is disabled.

Option Name **Number of GIF Things**  
CDL Name CYGPKG\_NET\_NGIF  
Description Sets the number of GIFs. The GIF code allows tunneling of different versions (4 or 6) of IP packets. The default value is 0.

Option Name **Number of Loopback Interfaces**  
CDL Name CYGPKG\_NET\_NLOOP  
Description Sets the number of loopback interfaces, which allows reception of outgoing packets. The default value for this option is 1.



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Error and Warning Log Control</b>                                                                                                                                                                                                                                                                                                                                                                                                                   |
| CDL Name    | CYGPKG_NET_XXX_LOGGING                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | Controls the type and amount of error/warning message output by the networking code. A 32-bit hexadecimal number is set to control which messages are output. This option is enabled by default.                                                                                                                                                                                                                                                       |
| Option Name | <b>Memory Designated for Networking Buffers</b>                                                                                                                                                                                                                                                                                                                                                                                                        |
| CDL Name    | CYGPKG_NET_MEM_USAGE                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description | Controls the amount of memory allocated for buffers used by the networking code. These buffers are used to hold the incoming and outgoing data. The default value is 256 kbytes.                                                                                                                                                                                                                                                                       |
| Option Name | <b>Max Number of Open Sockets</b>                                                                                                                                                                                                                                                                                                                                                                                                                      |
| CDL Name    | CYGPKG_NET_MAXSOCKETS                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Description | Controls the maximum number of network sockets that can be simultaneously opened. The default is 16.                                                                                                                                                                                                                                                                                                                                                   |
| Option Name | <b>Number of Supported Pending Network Events</b>                                                                                                                                                                                                                                                                                                                                                                                                      |
| CDL Name    | CYGPKG_NET_NUM_WAKEUP_EVENTS                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Description | Sets the total possible number of pending network events. An example of a pending network event is a <code>connect</code> function call waiting to complete. The default value is 8.                                                                                                                                                                                                                                                                   |
| Option Name | <b>Priority Level for Background Network Processing</b>                                                                                                                                                                                                                                                                                                                                                                                                |
| CDL Name    | CYGPKG_NET_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Description | Sets the priority level of the background networking thread. The default priority level is 7.                                                                                                                                                                                                                                                                                                                                                          |
| Option Name | <b>Priority Level for Fast Network Processing</b>                                                                                                                                                                                                                                                                                                                                                                                                      |
| CDL Name    | CYGPKG_NET_FAST_THREAD_PRIORITY                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | Sets the priority level of the fast network thread. This priority level should be higher than the background networking thread. The default priority level is 6.                                                                                                                                                                                                                                                                                       |
| Option Name | <b>Fast Network Processing Thread ‘Tickles’ Driver</b>                                                                                                                                                                                                                                                                                                                                                                                                 |
| CDL Name    | CYGPKG_NET_FAST_THREAD_TICKLE_DEVS                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description | Allows the fast network thread to unblock a network driver due to a hardware problem such as a lost interrupt. Attempting to send a packet allows the driver to become unblocked. This option is not necessary for protocols that exchange keep-alive packets periodically, such as TCP. A suboption sets the delay (in kernel ticks) to wait before sending the packet. The default is enabled with a delay of 50 kernel ticks (approximately 500ms). |
| Option Name | <b>Build Networking Tests (Demo Programs)</b>                                                                                                                                                                                                                                                                                                                                                                                                          |
| CDL Name    | CYGPKG_NET_BUILD_TESTS                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | Enables network tests to be built. This option is disabled by default.                                                                                                                                                                                                                                                                                                                                                                                 |

Some of the configuration options deserve a closer look because of their impact on the operation of the networking package. The configuration option *Memory Designated for Networking Buffers* is used to set up the memory used by the network stack. This memory consists of a number of memory buffers (mbuf), memory clusters, and a memory pool. The

mbuf has a fixed size of 128 bytes, including data and header information, which can store smaller-sized packets. If the packet exceeds the mbuf limit, a memory cluster, which has a fixed size of 4096 bytes, is added to contain the remaining data. The memory pool is like a private heap used by various functions of the network stack, which can malloc and free memory as needed. Fine-tuning the amount of memory designated for use by the Networking package might be necessary to handle the network data flow for the embedded device's specific application.

A good source for a detailed description of the memory architecture used in the 4.4BSD operating system, which is the basis for the networking packages, is the book *The Design and Implementation of the 4.4BSD Operating System* by Marshall Kirk McKusick and Keith Bostic.

Another configuration option we need to take a closer look at is the *Initialization Options for 'eth0'*. The Networking package contains configuration components for two Ethernet network interface devices. The second Ethernet network device is configured with the *Initialization Options for 'eth1'* configuration option. It is important to put some forethought into how each device is used in the released system; perhaps one interface can be used for debug and one for monitor and control via SNMP. It is necessary to ensure that each device is properly initialized with an appropriate network address when the system is initialized.

A MAC address needs to be preconfigured for each Ethernet network interface. The MAC address typically resides in some form of read-only memory that is set during hardware manufacturing. For some Ethernet drivers, the MAC address can be set by configuration data in the RedBoot ROM monitor or statically configured in the application.

There are three options available for the initialization of the Ethernet network interfaces. The first option is *Initialize 'eth0' Manually* (CYGHW\_NET\_DRIVER\_ETH0\_MANUAL). This requires the application to perform all device interface initialization and network address configuration to get the network interface up and running.

The next option we want to better understand is *Use BOOTP/DHCP To Initialize 'eth0'* (CYGHW\_NET\_DRIVER\_ETH0\_BOOTP). The suboption *Use DHCP Rather Than BOOTP for 'eth0'* (CYGHW\_NET\_DRIVER\_ETH0\_DHCP) determines what protocol is used for the network interface initialization. The default is to use DHCP; in which case, a DHCP server is needed to provide the IP address. When using BOOTP or DHCP for network interface initialization, the application needs to call the function `init_all_network_interfaces` to initialize the network interface using the selected protocol. The application must include the file `network.h` in order to use the networking package API functions.

If the configuration option *Use Full DHCP Instead of BOOTP* (CYGPKG\_NET\_DHCP) is enabled, a DHCP management thread is started, which monitors the network interface to ensure that the DHCP leases are properly handled. If this configuration option is disabled, the application is responsible for renewing DHCP leases. A semaphore, `dhcp_needs_attention`, and a function, `dhcp_bind`, are provided for the application to manage the DHCP leases.

The last option is *Address Setups For 'eth0'* (`CYGHWR_NET_DRIVER_ETH0_ADDRS`), which allows you to set static addresses for the network interface. The suboptions allow configuration of the IP address, network mask, broadcast address, and gateway address. Configuring static addresses can be useful in debugging scenarios; however, in a release system that runs the same application image it can be catastrophic to have multiple devices with the same IP address on the same network.

---

**NOTE** Debugging via the network interface is allowed by the RedBoot ROM monitor, see Chapter 9 for more details about RedBoot. RedBoot requires its own IP address for communication with the GDB host. The application needs to have a different IP address for its network communications. One solution is to statically configure the IP address RedBoot uses and have the application use DHCP or BOOTP to get its own IP address.

### 8.6.6 Networking Tests

The eCos networking package provides tests for exercising different features of network communications. The tests can be performed over any of the network interfaces. The network tests, located in the `tests` directory under the `common` networking directory, are also a good source for examples on performing various network functions. For example, the file `udp_lo_test.c` demonstrates how to initialize the network interface, create a UDP client for sending data, and create a UDP server for receiving data.

There are two configuration options for building network tests, one under the Basic Networking Framework package and the other under the specific networking stack implementation (OpenBSD or FreeBSD) package. Both configuration options are *Build Networking Tests (Demo Programs)* (`CYGPKG_NET_BUILD_TESTS`).

Some of the tests require additional host machines running programs included in the networking package tests. A makefile, `make.linux`, is included to build these tests on the host machine. Table 8.1 describes the tests available in the eCos Networking package.

**Table 8.1** Networking Tests

| Test Filename            | Description                                                                          |
|--------------------------|--------------------------------------------------------------------------------------|
| <code>bridge.c</code>    | Example network bridge application.                                                  |
| <code>dhcp_test.c</code> | Performs DHCP test by renewing lease and performing pings to the host.               |
| <code>flood.c</code>     | Continuously sends multiple ping packets, then waits for reception of the responses. |

**Table 8.1** Networking Tests *(Continued)*

| Test Filename                                                   | Description                                                                                                                                                                                                                                             |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ftp_test.c</code>                                         | Connects to an FTP server, then reads and writes data.                                                                                                                                                                                                  |
| <code>ga_server_test.c</code>                                   | Waits for a connection on a specified port, then retrieves a packet and closes the connection. This test uses <code>getaddrinfo</code> to set up its addresses.                                                                                         |
| <code>ipv6_server_test.c</code>                                 | Uses IPv6 and waits for a connection on a specified port, then retrieves a packet and closes the connection. A host running a telnet session is needed for this test.                                                                                   |
| <code>mbuf_test.c</code>                                        | Memory buffer (mbuf) allocation.                                                                                                                                                                                                                        |
| <code>multi_lo_select.c</code>                                  | Performs multiple select operations on different sockets for sending data.                                                                                                                                                                              |
| <code>nc_test_master.c</code><br><code>nc_test_slave.c</code>   | Characterizes the performance of a network by running the <code>nc_test_slave.c</code> on the target machine and the <code>nc_test_master.c</code> on the host machine. UDP and TCP are used to transfer data across the network.                       |
| <code>nc6_test_master.c</code><br><code>nc6_test_slave.c</code> | Characterizes the performance of a network using IPv4 and IPv6 by running the <code>nc6_test_slave.c</code> on the target machine and the <code>nc6_test_master.c</code> on the host machine. UDP and TCP are used to transfer data across the network. |
| <code>ping_test.c</code>                                        | Sends multiple ping packets to a server machine to verify communication, as well as to a nonexistent machine to verify the timeout of ping packets.                                                                                                     |
| <code>ping_lo_test.c</code>                                     | Ping test of the loopback address.                                                                                                                                                                                                                      |
| <code>server_test.c</code>                                      | Waits for a connection on a specified port, then retrieves a packet and closes the connection. A host running a telnet session is needed for this test.                                                                                                 |
| <code>set_mac_address.c</code>                                  | Sets the MAC address for the Ethernet network interfaces. All Ethernet drivers might not support this feature.                                                                                                                                          |
| <code>socket_test.c</code>                                      | Performs basic socket creation and setup.                                                                                                                                                                                                               |

**Table 8.1** Networking Tests (Continued)

| Test Filename                            | Description                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tcp_echo.c<br>tcp_sink.c<br>tcp_source.c | Data forwarding test to verify network performance. The file <code>tcp_source.c</code> runs on a host and sends data to the target machine. The file <code>tcp_echo.c</code> runs on the target, which receives the data and forwards it. The file <code>tcp_sink.c</code> runs on another machine and receives the forwarded traffic from the target.                                                               |
| tcp_lo_select.c                          | Performs a TCP throughput test using the loopback address using a server and two clients.                                                                                                                                                                                                                                                                                                                            |
| tcp_lo_test.c                            | Sets up a client and server to perform basic data transfer operations using TCP.                                                                                                                                                                                                                                                                                                                                     |
| tftp_client_test.c                       | Performs TFTP get and put operations using a host server. The host server needs to be configured independently of the target running the client program.                                                                                                                                                                                                                                                             |
| tftp_server_test.c                       | Runs a TFTP server on the target machine. Another machine running an TFTP client can be used to perform various operations. The server program runs for five minutes before closing down. The dummy file system used with this test contains one file, see <code>tftp_dummy_file.c</code> , and can accommodate the creation of three additional files up to 1MB in size. The filenames can be up to 256 bytes long. |
| udp_lo_test.c                            | Uses a client and server to transfer UDP data via the loopback address.                                                                                                                                                                                                                                                                                                                                              |

### 8.6.7 DNS Support

*Domain Name System* (DNS) is a standard that allows resolution of domain names into IP addresses. For example, instead of typing in 209.249.29.67 into your browser to visit a particular site, you can just type in `sources.redhat.com` and your browser is directed to the appropriate Web site. It is easier to remember the domain name rather than an IP address. DNS handles the translation from domain name to IP address.

The *DNS Client* (`CYGPKG_NS_DNS`) package is included by default when either the `net` or `new_net` templates are used. Otherwise, the package can be added separately using the eCos configuration tools. The DNS client source code is located in the `dns` subdirectory under the `net/ns` subdirectory.

A few restrictions exist for the DNS client. First, the DNS client is only supported for IPv4, not IPv6. Next, if the DNS server returns multiple records for a host name, the `hostent` only contains the record for the first entry. Finally, the `gethostbyname` and `gethostbyaddr`

functions are thread safe, allowing multiple threads to call these functions. However, it is not safe for a single thread to call both functions; this destroys the results from the previous call.

The DNS client is initialized by calling the function `cyg_dns_res_init`, located in the source file `dns.c`. This function takes the address of the DNS server that the client can use to query addresses for translation. The client understands that the target is in a domain, which is not used by default. The domain name is set using the function `setdomainname`.

## 8.7 SNMP Support

SNMP is the standard for the exchange of management information between network devices. SNMP uses simple requests and responses to communicate management information about a particular device on a network. A management application runs on a remote system that controls the SNMP agent, which runs on the device. SNMP uses UDP at the transmission layer. A Management Information Base (MIB) contains information and statistics for each device in a network. The MIB information is constantly updated to keep track of the current status of the networked device. The SNMP agent accesses the MIB information to fulfill the remote management requests.

The eCos SNMP package is a port of the UCD-SNMP version 4.1.2 code base. The UCD-SNMP implementations were done at Carnegie Mellon University and University of California at Davis. This code base has been transformed into Net-SNMP, which is maintained by SourceForge. Additional information about the original SNMP code base can be found online at:

<http://net-snmp.sourceforge.net>

This site also contains additional SNMP tools, including a tool to request or set information from SNMP agents, a tool to generate and handle SNMP traps, and a graphical Perl/Tk/SNMP-based MIB browser.

eCos SNMP support is contained in two packages, the agent and library. The SNMP package is arranged in this manner to remain consistent with the original UCD-SNMP implementation. The separation of packages also allows an eCos client application to use the SNMP library without including the agent package.

The agent contains application-specific handler files that allow a remote SNMP manager to retrieve the MIB information about the device. The library contains the code for formatting packets with the SNMP protocol and the MIB file parser, which uses Abstract Syntax Notation One (ASN-1). The library supports SNMP version 2 and contains security and authentication aspects of version 3. The database information is MIB-II compatible.

The eCos SNMP packages are located in the `snmp` subdirectory under the `net` subdirectory. The *SNMP Agent* (`CYGPKG_SNMPAGENT`) package is contained in the `agent` subdirectory, and the *SNMP Library* (`CYGPKG_SNMPLIB`) package is in the `lib` subdirectory. The SNMP package requires the inclusion of the Networking package.

The agent contains a single thread, `snmpd` located in the file `snmpd.c`, which is initialized in the routine `cyg_net_snmp_init`. The task runs at a priority level that is one lower than the background network task (`CYGPKG_NET_THREAD_PRIORITY`). The application must call the `cyg_net_snmp_init` function, located in the file `snmptask.c`, to start the SNMP agent.

The library package contains functionality that relies on the presence of a file system. Currently, this is not implemented in eCos; therefore, it is the responsibility of the application to supply file system support if this functionality is desired.

A MIB compiler utility is also included in the SNMP package. The MIB compiler is located in the `utils\mib2c` subdirectory within the agent package. The utility is a Perl script that takes the MIB, included in the SNMP agent package, and compiles it into C code. The output from the MIB compiler can then be modified to support the needed MIB functionality of a specific application. The new MIB can then be recompiled and used in the application. The default eCos MIB is located in the `mibgroup\mibII` subdirectory within the agent package. There are two `readme` files, in the `mib2c` subdirectory, which give additional information about the MIB compiler. These files are `readme.mib2c` and `readme-ecos`.

## 8.8 The GoAhead Embedded WebServer

Although SNMP is the long-time standard for remote management, more and more devices are seeking an alternative method to eliminate the shortcomings of SNMP. The latest trend is to use Web protocols, such as HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), and Java, for remote device management, also called Web-based management. This is accomplished through the use of an embedded Web server running on the device. A device's data can be used to generate dynamic content that is represented in a Web page. These pages are accessed by any standard browser and become the graphical interface to the device. Devices become more user friendly when they can be controlled using a common interface, such as the browser. For example, programming a VCR might not be as daunting a task if the job could be done via a browser interface.<sup>1</sup>

There are a few disadvantages to using SNMP for remote management. One disadvantage is the requirement of application software to control remote devices that is often costly and difficult to use. Another disadvantage is that SNMP uses UDP, which is an unreliable protocol, for communication across networks. Packets that are lost in transmission are not retransmitted, and acknowledgment of received packets is not performed. This can have dire consequences if the information contained in that packet was crucial to the system.

Along with being the hip, up-and-coming trend, Web-based management allows the use of a standard interface that is familiar to most users. Having the browser as an interface also allows the use of handheld devices, most of which have browsers as standard software, for portable management. Network communication from devices is accomplished using TCP, which is a reliable protocol.

Companies currently using the GoAhead WebServer in products include Hewlett-Packard, Honeywell, Siemens, and Canon. A complete list of companies is provided on the GoAhead WebServer site at:

[www.goahead.com/webserver/customers.htm](http://www.goahead.com/webserver/customers.htm)

<sup>1</sup> My wife, who finds programming the VCR a daunting task, can verify this and, therefore, relies on me for all of her video recording needs. Yet she is a wizard using a computer to navigate the Web with her favorite browser.

One disadvantage to the Web-based management approach is that once a page is served to the browser, the content is static and does not change until there is some user intervention, such as a refresh request. This can be a problem if a constant flow of data is needed or if a user needs to be notified of an alarm condition on the device. One solution to this problem is the use of the HTTP REFRESH tag, forcing the browser to re-request the data at a specified interval. Another solution is through the use of Java applets and JavaScript to continually request data from the device.

The GoAhead WebServer is an open-source embedded Web server specifically designed for use in embedded systems. The source code is written in C. Unlike typical server-based Web servers, the GoAhead WebServer is focused on meeting constraints found in an embedded system, including:

- Small memory footprint
- Configurable security model
- Supporting the generation of dynamic Web page content
- Support for devices that do not have a file system
- Portability across a wide range of platforms and CPU architectures
- Integration of the source code into very customized devices

The WebServer requires a TCP/IP stack and approximately 60 kbytes of memory. The GoAhead WebServer supports:

- Active Server Pages (ASP)
- In-process Common Gateway Interface (CGI)
- Embedded JavaScript
- HTTP 1.0 with persistent connections found in HTTP 1.1
- 65 connections per second
- Secure Sockets Layer (SSL) version 3.0
- Digest Access Authentication (DAA)
- User Management via login access
- Storage of Web pages in ROM

Currently, SSL support is not provided for in eCos. Additional information can be found online at the GoAhead WebServer site at:

[www.goahead.com/webserver/webserver.htm](http://www.goahead.com/webserver/webserver.htm)

Similar to eCos licensing, the GoAhead WebServer offers its source code free of charge in exchange for any enhancements to the source code base. There are three basic requirements to using the GoAhead WebServer in a product. First, GoAhead must be notified prior to shipping the product using the WebServer. Second, the GoAhead mark (which can be found on their Web site) must be displayed on the initial page. Finally, GoAhead is allowed to identify companies using the WebServer for marketing efforts. The GoAhead WebServer license can be found online at:



[www.goahead.com/webserver/license.htm](http://www.goahead.com/webserver/license.htm)

The GoAhead site contains a searchable database of questions, and their solutions. GoAhead offers a bug report form that allows you to submit specific problems with the Web server. Bugs can also be emailed to:

[webserverbugs@goahead.com](mailto:webserverbugs@goahead.com).

There is also a newsgroup devoted to the GoAhead WebServer, which does not get a large volume of traffic. The newsgroup can be found at:

<news://news.goahead.com/goahead.public.webserver>

## 8.9 Symmetric Multi-Processing Support

SMP is a computer architecture that uses multiple CPUs to process program code. The multiple CPUs share a common operating system and memory subsystem. This allows the processors to work together to share the workload in an embedded system, which provides higher performance than a single-processor system. eCos provides SMP support on selected architectures and platforms. This support is broken down into HAL- and kernel-level support. SMP support is only available in the multilevel queue scheduler.

eCos does impose some target hardware limitations in its SMP support, including:

- The maximum number of CPUs supported is eight, with the typical number being two or four.
- The hardware must supply a synchronization mechanism in the form of a test-and-set or compare-and-swap instruction. The eCos kernel uses these hardware instructions for the spinlock implementation.
- No caches are used, all processors share the cache in the system, or the hardware maintains coherent caches. This prevents eCos from performing cache flush operations around each memory access.
- All memory shared among CPUs is addressed at the same location for all CPUs.
- All devices are accessible by all CPUs.
- An interrupt controller must be present to route interrupts to a specific CPU. Other acceptable architectures include all interrupts delivered to a single CPU, certain interrupts bound to a specific CPU, or certain interrupts local to each CPU. eCos does not support delivering all interrupts to all CPUs in the system and allowing the software to resolve conflicts.
- To allow events on one CPU to cause rescheduling on another CPU, a mechanism is needed to allow one CPU in the system to interrupt another CPU.
- Software that is running on a particular CPU must be able to identify which CPU it is running on.

The startup sequence is different on SMP systems. A single CPU, called the primary, handles the startup initialization sequence, while the other CPUs, called secondary, are either placed in a suspended state or put into an idle loop by the HAL. After the application calls `cyg_scheduler_start`, the secondary CPUs are initialized.

One major issue in an SMP system is the ability for the kernel to protect its data structures from concurrent accesses. This is easily accomplished in a single-processor system by disabling interrupts. However, in a system with multiple CPUs, disabling interrupts on one CPU does not block another CPU from accessing the data structures.

When the kernel is operating in a single-processor system, accesses to the kernel data structures are blocked using the scheduler lock. The scheduler lock is implemented as a simple counter that is atomically incremented to acquire the lock and decremented to release the lock. When the value of the lock counter is zero, the scheduler is allowed to select a different thread to run. Since ISRs can be serviced while the scheduler is locked, certain kernel functions can only be called from certain operating contexts. Additional information about the scheduler for single-processor systems can be found in *The Scheduler* section of Chapter 5, *The Kernel*.

For SMP systems, a kernel-locking mechanism is needed that does not rely on interrupt manipulation. This special synchronization mechanism is called a spinlock. Spinlocks are provided for SMP systems, although the other synchronization mechanisms work in SMP systems as well. Spinlocks are covered in Chapter 6, *Threads and Synchronization Mechanisms*.

In addition, functions have been added for interrupt processing in SMP systems. One of these functions allows routing of interrupts to a specific CPU in the system. The other function enables you to find out which interrupts in the system are handled by which CPUs. Detailed information about these, and other interrupt function calls can be found in Chapter 3, *Exceptions and Interrupts*.

## 8.10 Additional Features

Now that we have covered some of the major additional functionality available for use in the eCos system, let's take a quick look at some other additional features supported by eCos. This section is intended to make you aware of some of the other features supported by eCos; however, the details of operation for these packages are omitted.

The dynamic loader allows Executable and Linking Format (ELF) file images to be loaded onto a target during run time. After the new ELF image is loaded, the running application can call functions located in the new image. The dynamic loader package (`CYGPKG_LOADER`) is located in the `services/loader` directory.

The zlib package is a general-purpose compression/decompression library widely known in the software community. It provides in-memory compression and decompression functions that are thread safe and include integrity checks of the uncompressed data. Reading and writing of files in gzip format, with interface functions similar to `stdio`, is also supported by the zlib library. The zlib package (`CYGPKG_COMPRESS_ZLIB`) contains a port of zlib to eCos and is contained in the `services/compress/zlib` directory. Additional information about the zlib library can be found online at:

[www.zip.org/zlib](http://www.zip.org/zlib)

eCos provides support for the Microwindows Graphical User Interface (GUI). Microwindows is an open-source project focused on allowing the features of modern graphical windowing interfaces to be run on smaller devices. The Microwindows package (`CYGPKG_MICROWINDOWS`) is located under the `services\gfx\mw` directory. Additional information about Microwindows is provided under the `doc` directory within the package as well as online at:

[www.microwindows.org](http://www.microwindows.org)

A generic power management package is also provided by eCos. This package provides a framework that allows the incorporation of additional power management facilities in an embedded system. The power management package (`CYGPKG_POWER`) is located in the `services\power` directory. Additional information about the power management package is contained in the `doc` directory within the package.

## 8.11 Summary

In this chapter, we took a brief look at some of the additional features provided in the eCos system and by third-party contributors that can be included within an application. These additional features extend eCos' core functionality, allowing eCos to meet the requirements of a wider range of embedded systems. Now that you are aware of these features, with some additional investigation they can quickly be incorporated into your system.

## References

- Sakamura, Ken. *μITRON 3.0, An Open and Portable Real-Time Operating System for Embedded Systems*. (IEEE Computer Society Press, 1997).
- Stevens, W. Richard. *TCP/IP Illustrated, Volume 1: The Protocols*. (Addison-Wesley, 1994).
- McKusick, Marshall Kirk, and Keith Bostic. *The Design and Implementation of the 4.4BSD Operating System*. (Addison-Wesley Longman, Inc., 1996).

# The RedBoot ROM Monitor

**T**his chapter covers the RedBoot ROM monitor embedded software tool. The RedBoot ROM monitor provides debugging and bootstrap support. We cover the installation and configuration details necessary to get RedBoot running on target hardware and communicating with a host unit.

We look at the RedBoot user interface and commands provided for controlling the features provided. It is important to understand the method for building a RedBoot image to make use of upgrades and to extend the default command set; however, we cover this information in later chapters of this book. Although RedBoot is a standalone program that can be used with any real-time operating system, the information in this chapter focuses on using RedBoot with eCos applications.

## 9.1 Overview

RedBoot is an acronym for Red Hat Embedded Debug and Bootstrap. It is a program designed for embedded systems to provide a debugging and bootstrap environment. RedBoot is intended to take the place of older programs; specifically, CygMon and GDB stub ROM. In fact, CygMon is no longer supported.

RedBoot is an eCos-based application and uses the eCos Hardware Abstraction Layer for its foundation. However, RedBoot can be used on any embedded system or with any RTOS. RedBoot can be used for debug support during the product development cycle or in a released product to provide flash and network booting. Some of the features provided by RedBoot include:

- Boot scripting support
- Command Line Interface (CLI) for monitor and control support

- Access via serial or Ethernet ports
- GDB support
- Flash image system support
- X/Y modem support
- Network bootstrap support using BOOTP or static IP address configurations

RedBoot is a standalone, self-contained module that runs as an application on the target platform. Included in RedBoot is a GDB stub (see Chapter 8, *Additional Functionality and Third-Party Contributions*, for more information) that allows connection to the target platform from a GDB host for application debugging. This debug connection can be over the serial or Ethernet port. RedBoot uses a standalone stack, separate from the eCos networking stack, which provides a base of functionality for network communications.

RedBoot supports booting from ROM, RAM, floppy disk, or an Integrated Disk Electronics (IDE) hard drive running the ext2 file system. A ROMRAM startup mode is also available; in which case, RedBoot resides in ROM but is copied to RAM before it begins to run.

When we build a RedBoot image, as in Chapter 12, *An Example Application Using eCos*, the result is a binary file that is programmed directly onto the target hardware. This RedBoot image can contain, depending on the configuration options and packages, all the software components to use all features of the target hardware.

Figure 9.1 shows the block diagram architecture of some of the features included with the RedBoot ROM monitor. From this illustration, we can see that the functionality provided by RedBoot is all contained within the RedBoot program image. The interaction between RedBoot and the eCos application varies depending on the configuration option settings in both images. RedBoot can be configured to provide simple load and run functionality, or can be configured to provide flash management support, monitor and control for the eCos application via the CLI using the serial port, and GDB debugging capabilities.

RedBoot is shipped in many products currently on the market, including:

- Intel Residential Gateway
- Intel XScale Development Board
- Intel StrongARM Development Boards
- MIPS Malta 4kc/5kc/20kc Development Board
- MIPS Atlas 4kc/5kc/20kc Development Board

The latest developments and information about the RedBoot ROM monitor can be found online at:

<http://sources.redhat.com/redboot>

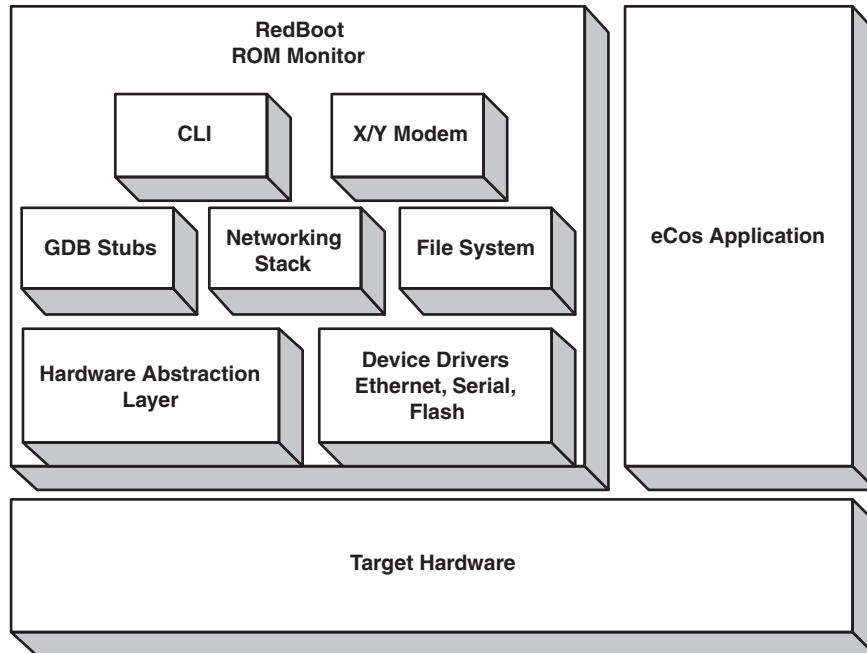


Figure 9.1 RedBoot ROM monitor architecture.

## 9.2 RedBoot Directory Structure

Although RedBoot is a standalone application, which can run on a target system with or without eCos, the directory structure of RedBoot is similar to other eCos packages. Figure 9.2 shows the RedBoot directory structure.

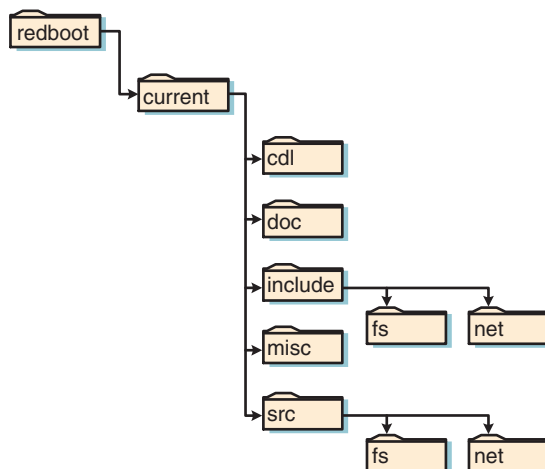


Figure 9.2 RedBoot directory structure.

As we see in Figure 9.2, the RedBoot package is contained in the directory `redboot`. Additional details about the typical eCos package directory structure are covered in Chapter 11, *The eCos Toolset*.

Beneath the main directory is the version of RedBoot, which might vary depending on the source code used. However, in our installation, the version directory is named `current`, as we see in Chapter 10, *The Host Development Platform*.

The `cdl` directory contains the CDL description file of the RedBoot package. The CDL file contains the information needed for RedBoot configuration options, packages, and build information.

Next is the `doc` directory. This contains documentation about the RedBoot ROM monitor for use by the eCos configuration tools.

The `include` directory contains the source code header files for RedBoot. This directory contains two additional directories. The `fs` directory contains the RedBoot file system header files, and the `net` directory contains the RedBoot networking header files.

The `misc` directory contains an eCos minimal configuration file for RedBoot. Finally, the `src` directory contains all of the RedBoot source code. The file system source code is contained under the `fs` directory, and the networking code is contained under the `net` directory.

### 9.3 Installation and Configuration

eCos provides a template called `Redboot`. Using this template, the default packages used with the RedBoot ROM monitor are loaded and configured with the default option settings. The specific packages loaded depend on the hardware platform you are using. For example, the Motorola PowerPC MBX860 board platform loads the Ethernet device driver since this is supported on the MBX platform. However, the Intel StrongARM SA-1100 Multimedia board platform does not contain Ethernet support; therefore, the Ethernet device driver is not loaded on the SA-1100 Multimedia platform. We cover building a RedBoot image in Chapter 12.

---

**NOTE** Most HAL packages include a `.ecm` file that can be imported, along with using the RedBoot template, in order to set up the packages needed to build the RedBoot ROM monitor for a specific platform. The files are typically named `redboot_RAM.ecm`, `redboot_ROM.ecm`, or `redboot_ROMRAM.ecm`, where ROM, RAM, and ROMRAM determine the startup type for the RedBoot ROM monitor. The files are located in the `misc` subdirectory under the HAL packages. We cover the details of using `.ecm` files with the Configuration Tool in Chapter 11.

Although `.ecm` files are useful for loading a baseline of the proper packages for a specific platform, you should still verify the configuration option settings prior to building the RedBoot image. This ensures that the proper functionality you need is supported by RedBoot; for example, the configuration of certain options allows RedBoot to include certain required functionality, or alternatively eliminate particular features, thus reducing RedBoot's memory footprint.

RedBoot typically resides in the flash boot sector or boot ROM on the target platform. This gives control to RedBoot when the board is powered up. RedBoot can be configured to run from RAM if you need to debug the RedBoot code itself. This might be the case if you are porting RedBoot to run on your own hardware platform.

The method for programming the RedBoot image onto the target hardware varies from platform to platform and depends on the tools available. Some platforms include their own ROM monitor, such as the Motorola PowerPC MBX860 board, which can be used to program an image into flash memory and then run the programmed image on power up. In other cases, a device programmer might need to be used in order to burn the image into memory. Detailed instructions on programming a RedBoot image for each target platform supported by RedBoot can be found online at:

<http://sources.redhat.com/ecos/docs.html>

The steps involved in running an image on the i386 PC platform and the Motorola PowerPC MBX860 platform are covered in Chapter 12. These same steps can be applied to running the RedBoot image as well.

The resources (ROM, RAM, and communication ports) used by RedBoot vary for each platform. A description of the resource utilization for the supported platforms can be found in the RedBoot online documentation. The Configuration Tool's Memory Layout viewer can also be used to check and modify the RAM and ROM resources used by RedBoot. We cover the Configuration Tool, and all of its features, in Chapter 11.

### 9.3.1 RedBoot Configuration

The configuration of the RedBoot image is dependent on the features you want provided by the ROM monitor and the features provided in the application. As previously mentioned, the RedBoot ROM monitor is a standalone program and should be viewed as completely independent and separate from your application. However, the steps for configuring and building a RedBoot image are the same as the steps for configuring and building your application. The configuration allows you to control the functionality provided by RedBoot in order to meet the resource requirements of your system.

The *RedBoot ROM Monitor* (CYGPKG\_REDBOOT) package is contained in the `redboot` subdirectory under the eCos repository root directory. Item List 9.1 shows the configuration options available for the RedBoot ROM monitor. Some of the options specified in the table are not supported by all platforms and, therefore, cannot be configured.

#### Item List 9.1 RedBoot Configuration Options

|             |                                                          |
|-------------|----------------------------------------------------------|
| Option Name | <b>Include Support for ELF File Format</b>               |
| CDL Name    | CYGSEM_REDBOOT_ELF                                       |
| Description | Allows application files in ELF to be loaded by RedBoot. |



|             |                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Build RedBoot ROM ELF Image</b>                                                                                                                                                                                                                                                                                                                                    |
| CDL Name    | CYGBLD_BUILD_REDBOOT                                                                                                                                                                                                                                                                                                                                                  |
| Description | Builds an ELF of the RedBoot image. This option is disabled by default. The suboptions under this option allow inclusion of decompression support, thread debugging support, and setting a customized version string.                                                                                                                                                 |
| Option Name | <b>RedBoot Networking</b>                                                                                                                                                                                                                                                                                                                                             |
| CDL Name    | CYGPKG_REDBOOT_NETWORKING                                                                                                                                                                                                                                                                                                                                             |
| Description | Contains suboptions for configuring network support. The default mode for this option is enabled for platforms with Ethernet support. The suboptions allow you to configure a default IP address when BOOTP is used (disabled by default), specify the TCP port for incoming connections (default is 9000), and specify the number of network buffers (default is 4). |
| Option Name | <b>Allow RedBoot to Use Any I/O Channel for Console</b>                                                                                                                                                                                                                                                                                                               |
| CDL Name    | CYGPKG_REDBOOT_ANY_CONSOLE                                                                                                                                                                                                                                                                                                                                            |
| Description | RedBoot attempts to use all serial I/O channels that are defined for console communication. When input arrives, RedBoot uses that channel for its console. This option is enabled by default.                                                                                                                                                                         |
| Option Name | <b>Allow RedBoot to Adjust the Baud Rate Serial Console</b>                                                                                                                                                                                                                                                                                                           |
| CDL Name    | CYGSEM_REDBOOT_VARIABLE_BAUD_RATE                                                                                                                                                                                                                                                                                                                                     |
| Description | Enables RedBoot to support baud rate set commands. This option is enabled by default.                                                                                                                                                                                                                                                                                 |
| Option Name | <b>Maximum Command Line Length</b>                                                                                                                                                                                                                                                                                                                                    |
| CDL Name    | CYGPKG_REDBOOT_MAX_CMD_LINE                                                                                                                                                                                                                                                                                                                                           |
| Description | Sets the maximum length for CLI commands. The default value is 256.                                                                                                                                                                                                                                                                                                   |
| Option Name | <b>Command Processing Idle Timeout (ms)</b>                                                                                                                                                                                                                                                                                                                           |
| CDL Name    | CYGNUM_REDBOOT_CLI_IDLE_TIMEOUT                                                                                                                                                                                                                                                                                                                                       |
| Description | Specifies the period before command processing is considered idle. A smaller value for this option causes more frequent idle processing. The default value is 10 ms.                                                                                                                                                                                                  |
| Option Name | <b>Size of ZLIB Decompression Buffer</b>                                                                                                                                                                                                                                                                                                                              |
| CDL Name    | CYGNUM_REDBOOT_LOAD_ZLIB_BUFFER                                                                                                                                                                                                                                                                                                                                       |
| Description | Sets the size (in bytes) of the buffer that is filled with incoming data during a load before calling the decompression function. The default value is 64 bytes.                                                                                                                                                                                                      |
| Option Name | <b>Validate RAM Addresses During Load</b>                                                                                                                                                                                                                                                                                                                             |
| CDL Name    | CYGSEM_REDBOOT_VALIDATE_USER_RAM_LOADS                                                                                                                                                                                                                                                                                                                                |
| Description | Allows RedBoot to check the validity of user RAM when loading an image. This option is enabled by default; care should be taken when disabling this option.                                                                                                                                                                                                           |
| Option Name | <b>Allow RedBoot to Support Flash Programming</b>                                                                                                                                                                                                                                                                                                                     |
| CDL Name    | CYGPKG_REDBOOT_FLASH                                                                                                                                                                                                                                                                                                                                                  |
| Description | Enables RedBoot to provide flash image system management. This option is enabled by default for platforms that contain flash support. Suboptions allow specific configuration of                                                                                                                                                                                      |

the flash memory, such as the minimum image size and the offset where the flash image system begins.

|             |                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Option Name | <b>Allow RedBoot to Support Disks</b>                                                                                                                                                                              |
| CDL Name    | CYGPKG_REDBOOT_DISK                                                                                                                                                                                                |
| Description | Enables RedBoot commands for loading disk files. This option is enabled by default for platforms with disk support. Suboptions allow specific configuration of the disk details such as number of supported disks. |
| Option Name | <b>Boot Scripting</b>                                                                                                                                                                                              |
| CDL Name    | CYGPKG_REDBOOT_BOOT_SCRIPT                                                                                                                                                                                         |
| Description | Component that contains the boot scripting configuration options. The options set the script timeout value and default boot script. The default is enabled.                                                        |
| Option Name | <b>Behave Like a ROM Monitor</b>                                                                                                                                                                                   |
| CDL Name    | CYGPRI_REDBOOT_ROM_MONITOR                                                                                                                                                                                         |
| Description | Allows RedBoot to provide ROM monitor services to applications that are loaded. This option is always enabled when building a ROM startup type image.                                                              |
| Option Name | <b>Allow RedBoot to Handle GNUPro Application ‘syscalls’</b>                                                                                                                                                       |
| CDL Name    | CYGSEM_REDBOOT_BSP_SYSCALLS                                                                                                                                                                                        |
| Description | RedBoot installs a syscall handler to support application debugging based on GNUPro newlib/bsp when this option is enabled. The default for this option is disabled.                                               |

The RedBoot image contains its own implementation of the HAL for the specific hardware platform selected. The HAL package is the same that is used in your application. The difference is how the HAL configuration options are set. The standard RedBoot configuration enables the RedBoot HAL option *Behave as a ROM Monitor*. Enabling this configuration option includes setting the *Startup Type* to ROM, which directs the HAL to perform all initialization necessary to bring up the hardware from a powered-off state. Using RedBoot in this standard configuration allows applications to be loaded into RAM for debugging on a hardware platform that is properly initialized.

Included in the HAL configuration is the setup of the virtual vectors. The standard configuration allows RedBoot to initialize the entire VVT and claim all of the virtual vectors in the table. The default configuration options for the VVT, which can be overridden, are that the ROM monitor provides the debugging and diagnostic I/O services and RAM applications rely on these services. Refer to Chapter 4, *Virtual Vectors*, for detailed information on virtual vectors and the VVT.

Because the VVT resides at a fixed area of memory, known by both the ROM monitor and the RAM application, the application can take over any services in the VVT at run time, leaving other services to be provided by RedBoot. RedBoot is capable of providing debugging and diagnostic services via the serial or Ethernet port. The ports used for debugging and diagnostic messages depend on the resources available on the particular platform and your debug configuration. For example, if your platform contains a serial port and an Ethernet port, you can use one port

for RedBoot debugging, and the other can be used by your application. With the proper virtual vector setup and configuration option settings, RedBoot is also able to share the ports it uses for debugging and diagnostics with the eCos application.

---

**NOTE** To avoid terminating a RedBoot communication channel that is using the networking stack, you must ensure proper configuration of two options within the eCos application image.

The first configuration option, *Inherit Console Settings From ROM Monitor* (CYGSEM\_HAL\_VIRTUAL\_VECTOR\_INHERIT\_CONSOLE), must be enabled. The second configuration option, *Claim Comms Virtual Vectors* (CYGSEM\_HAL\_VIRTUAL\_VECTOR\_CLAIM\_COMMS), must be disabled. These configuration options are located under the *ROM Monitor Support* component within the common configuration components of the eCos HAL.

One issue to keep in mind when configuring your debug environment settings is that using a serial port for debug communications can often be slow when you are loading large application images.

Typically, it is better if the port resources can be dedicated for either RedBoot usage or application usage. The HAL configuration options allow you to set up the debug and diagnostic I/O communications to meet your specific needs. The following two configuration options are used to configure the method of communication for the application. The first configuration option, *Route Diagnostic Output to Debug Channel* (CYGDBG\_HAL\_DIAG\_TO\_DEBUG\_CHAN), under the *Platform-Independent HAL Options* component within the common configuration components allows the use of RedBoot's debug channel for diagnostic messages from the application. RedBoot provides mangling support, using GDB, which then outputs the message to the host. Mangling in this scenario means that RedBoot converts diagnostic messages from the application into a properly formed GDB remote protocol packet.

The second configuration option is *Inherit Console Settings From ROM Monitor* (CYGSEM\_HAL\_VIRTUAL\_VECTOR\_INHERIT\_CONSOLE) under *ROM Monitor Support* within the common configuration components. This option allows the application to use the currently configured RedBoot console setup for output from the application. RedBoot again handles the mangling needed for proper communication.

Two platform-specific configuration options allow specific designation of the port number used for debug and diagnostic communications. If only one port exists on the platform, these options should be configured to use that port. The first is *Debug Serial Port* (CYGNUM\_HAL\_VIRTUAL\_VECTOR\_DEBUG\_CHANNEL), which determines the port used to connect to the GDB host.

The other configuration option is *Diagnostic Serial Port* (CYGNUM\_HAL\_VIRTUAL\_VECTOR\_CONSOLE\_CHANNEL), which determines the port used for output of diagnostic messages; for example, using the `diag_printf` function. Chapter 4 details these configuration

options and the steps involved for a typical debug environment configuration using the RedBoot ROM monitor with a separate application RAM image.

---

**NOTE** If the RedBoot image is built with the configuration option *Allow RedBoot To Use Any I/O Channel For Its Console* (CYGPKG\_REDBOOT\_ANY\_CONSOLE) enabled, which is the default value, the debug and diagnostic serial port configuration settings are ignored and RedBoot latches on to the first port that it receives input.

### 9.3.2 Host Configuration

Currently, there are two methods for connecting a host to a target running RedBoot, serial or Ethernet. With either method used, RedBoot uses the connection for both GDB and RedBoot command communications, although RedBoot can be configured to use separate communication channels for debugging and commands. RedBoot provides commands, which are detailed in the *User Interface and Command Set* section of this chapter, for configuring the different parameters associated with both communication methods. After setting these parameters, they are used on subsequent boots of the target hardware.

One important point to understand about the RedBoot driver implementations, for both serial and Ethernet, is that polling (not interrupts) is used for I/O communication. The configuration option *Command Processing Idle Timeout (ms)* (CYGNUM\_REDBOOT\_CLI\_IDLE\_TIMEOUT) sets the RedBoot command processing interval. After this timeout expires, RedBoot assumes the command console is idle and performs various background tasks. The default value for this option is 10 milliseconds. If traffic increases, there might be a point at which the RedBoot driver cannot keep up and packets are dropped. In that case, it might be necessary to adjust the timeout configuration option or re-check the traffic that RedBoot needs to process on the communication channel.

#### 9.3.2.1 Serial

Using the serial connection requires a host-side terminal program. The default parameters that RedBoot uses for a serial connection are specific to the target hardware platform. On some target hardware platforms, the RedBoot baud rate setting can be changed using the `baudrate` command, which is detailed in the *User Interface and Command Set* section of this chapter.

Most hardware target platforms should contain at least one serial port for RedBoot communications. Some platforms contain multiple serial ports. The serial port RedBoot uses for serial communication is described in the *RedBoot Configuration* section of this chapter. By default, RedBoot attempts to communicate with the host via the serial port, set by the configuration options, on power up. Once a command is received by RedBoot on one of the serial ports, that port is used for communications. There is a `channel` command available, which is detailed in the *User Interface and Command Set* section of this chapter, that allows the port RedBoot uses for console communication to be changed.

### 9.3.2.2 Ethernet

The other method for RedBoot communication is using an Ethernet port. In order to use an Ethernet port, the RedBoot image must include networking support and, obviously, the target hardware must have an Ethernet port.

One issue that must be resolved is the method for RedBoot to obtain the Ethernet MAC address. RedBoot attempts to use the MAC address provided by the board manufacturer. However, the MAC address is not provided on all target hardware platforms. On some hardware platforms, the Ethernet driver allows RedBoot to program a MAC address into the flash configuration, using the `fconfig` command. A default MAC address can also be built into the RedBoot image.

Another issue when using the Ethernet port for RedBoot communication is obtaining an IP address. RedBoot offers two methods for obtaining an IP address for network communication, static and dynamic. Selection of the method is controlled by the `fconfig` command, which is described in detail in the *User Interface and Command Set* section of this chapter.

When using the static method, you must set a unique IP address for the RedBoot image. Several IP address ranges are good to use because they are private addresses. By using private addresses, you can create a private network for your debugging configuration, and packets that leave this private network are not forwarded. According to RFC 1918, the reserved private addresses are:

- 10.0.0.0 to 10.255.255.255
- 172.16.0.0 to 172.31.255.255
- 192.168.0.0 to 192.168.255.255

It is important to realize that RedBoot does not offer any routing support; therefore, the IP address selected for the target hardware running RedBoot and the host must be on the same subnet. RedBoot uses ARP to resolve the host address and then sends packets directly to the host unit.

When using the dynamic method, the BOOTP protocol is enabled in the RedBoot image. Again, selecting the dynamic method is accomplished through the `fconfig` command. The dynamic method also requires a BOOTP server running on the host in order to provide an IP address to the target hardware. There are several free or shareware BOOTP servers available on the Web, including the BOOTP Turbo Server from Weird Solutions and a DHCP/BOOTP Server from Dr. Herbert Hanewinkel.

Once the IP address has been successfully set on the target hardware, you can use the ping utility to verify communication between the target and the host. Most host operating systems should offer this utility. RedBoot also contains a `ping` command, which is described in the *User Interface and Command Set* section of this chapter.

After the IP address is assigned to the target hardware Ethernet port, you can connect to RedBoot from the host using a Telnet client application. Most host operating systems provide a Telnet application utility and if not, again, a search on the Web should bring up a few options. Next, the port number that is used for Telnet protocol communications must be set on the host

---

**NOTE** When using the Ethernet port for RedBoot communication and in your application, it is important to set up the IP addresses properly. The system distinguishes the destination of a packet using the IP address. If the IP address for RedBoot and the application are the same, RedBoot will receive all incoming Ethernet packets, not passing them on to the application for processing. To avoid this situation, it is best to use a static IP address in one image and a separate static or dynamic image in the other image.

**NOTE** If you are using RedBoot for debugging purposes and it will not be shipped in the final product, you can assign a static IP address for RedBoot to use and allow the application to obtain its IP address via BOOTP or DHCP. This allows you to test the BOOTP/DHCP protocol in the application.

application and in RedBoot. By default, RedBoot uses port 9000. The default port can be changed using the `fconfig` command. Once the host connects to RedBoot using Telnet, all RedBoot communications take place over the Ethernet port.

---

**NOTE** If your application contains complex networking software, it is better to use a serial port for RedBoot communication and allow the application to use the Ethernet port exclusively. Although RedBoot and the application can share an Ethernet device, there are times when debugging is simplified if a separate communications channel is used. If the application image is large and you have concerns over the download time via a serial port, one solution is to download the application over the Ethernet port and then switch to the serial port for the debugging.

## 9.4 User Interface and Command Set

RedBoot provides a CLI. The port, serial or Ethernet, which the CLI is available on is determined by configuring RedBoot; refer to the *Host Configuration* section of this chapter. Once the initialization message is output by RedBoot, commands can be entered to control the target. An example of the RedBoot initialization message for the Intel x86 processor is shown in Code Listing 9.1.

```
1 Ethernet eth0: MAC address 00:d0:b7:92:9d:d2
2 IP: 192.168.0.10, Default server: 192.168.0.1, DNS server IP: 0.0.0.0
3
4 RedBoot(tm) bootstrap and debug environment [FLOPPY]
5 Non-certified release, version UNKNOWN - built 17:55:00, Apr 21 2002
6
7 Platform: PC (I386)
8 Copyright (C) 2000, 2001, 2002, Red Hat, Inc.
```

```
9
10 RAM: 0x00000000-0x000a0000, 0x00088870-0x000a0000 available
11 RedBoot>
```

**Code Listing 9.1** Example RedBoot initialization message.

In Code Listing 9.1, line 1 shows the Ethernet port (`eth0`) MAC address, which RedBoot retrieved from the Ethernet card. Line 2 displays the IP addresses for the target platform (192.168.0.10, which was statically configured in this case), the default server (192.168.0.1), and the DNS server, which has no IP address configured.

On line 4 we see the RedBoot title message along with the Startup Type configuration option selection; in this case, `FLOPPY`. The build information is output on line 5. In this case, the image was created from a snapshot of the RedBoot source code; therefore, it is not a certified release and there is no version number. The build time and date are also displayed, which occurred on April 21, 2002 at 5:55 P.M.

Line 7 displays the platform information showing this RedBoot image is for the i386 processor running on a PC platform. Line 8 gives the copyright information.

The RAM resource information is displayed on line 10. The addresses `0x0000_0000` to `0x000A_0000` show the memory used by RedBoot; in this case, 640 kbytes. Also shown, `0x0008_8870` to `0x000A_0000`, is the amount of memory available for loading programs or other development needs.

---

**NOTE** The i386 PC platform also has memory above `0x0010_0000` (1 Mbyte) and upward available for loading programs. RedBoot does not display this in the initialization message for the i386 PC platform. However, eCos RAM applications are loaded above the `0x0010_0000` address by default.

Finally, line 11 shows the RedBoot prompt, which indicates that RedBoot is ready to receive commands.

### 9.4.1 RedBoot Commands

The basic form of a RedBoot command takes the form:

```
RedBoot> COMMAND [-OPTION1] [-OPTION2 VALUE] OPERAND
```

where `COMMAND` is one of the supported RedBoot commands. The two optional switches, `OPTION1` and `OPTION2`, modify the behavior of a standard command. The `OPERAND` specifies additional information needed for particular commands.

Commands are not case sensitive and can be abbreviated to their shortest unique string. For example, to execute the `dump` command on a default (default means no other commands beginning with `d` have been added) RedBoot image, `d`, `du`, `dum`, or `dump` can be entered.

There are a few basic editing commands used in RedBoot:



- Delete (0x7F) and Backspace (0x08) erase the previous character.
- Period (.) (0x2E) halts the editing of a command, leaving the current item unchanged.
- Caret (^) (0x5E) is used as an up arrow for commands, such as `fconfig`, which go through a list of parameters to set.
- Dollar sign (\$) (0x24) switches RedBoot into GDB stub mode. The GDB stub takes control and waits for a connection from a GDB host. To get out of GDB stub mode, a disconnect message is sent from the GDB host or by a reset of the target.
- Return (0x0D) leaves the value for a particular parameter unchanged.

RedBoot generates basic error messages when commands are entered incorrectly or invalid commands are used. In the following example, the `dump` command is entered; however, the `-b` switch was omitted by mistake causing RedBoot to generate the error message we see on the following line. The message informs us that we have forgotten the switch for our parameter value `0x12`. RedBoot then outputs a new prompt following the error message for the next command.

```
RedBoot> dump 0x12
** Error: no default/non-flag arguments supported
RedBoot>
```

RedBoot commands are designed to be simple to use and remember. Although, the command set only offers basic commands, the source is available so you can add commands specifically needed for your platform. If you add general-purpose commands that you think are extremely useful, the eCos open-source community would be glad to accept a contribution.

Numbers entered with commands can be entered in decimal or hexadecimal (designated by the `0x` prefix). Item Lists 9.2 and 9.3 list the commands available in the default RedBoot image. Optional parameters or switches in the syntax for commands are enclosed in brackets (`[]`), and value parameters are enclosed in angled brackets (`<>`). Commands entered without parameters cause a help message about the command to be output or the current setting for the command.

All RedBoot commands listed in Item List 9.2 might not be available on all platforms because not all platforms support certain resources. For example, a platform that does not contain flash memory does not support the Flash Image System (FIS) command set. Item List 9.3 lists the FIS command set.

### Item List 9.2 RedBoot ROM Monitor Command Set

Syntax:        **alias** name [value]

Options:      name—String name for alias.  
              value—Expression for alias.

Description:    Allows a simple command-line alias to be used for longer expressions. When the pattern `%{name}` appears in a command line, it is replaced with the value previously set. Aliases can also be used in scripts. These aliases are kept in flash memory. The following shows an example using the alias command. On line 1 the alias `SBUF` is used for the string `-b 0x100000`. Therefore, any place `%{SBUF}` is used, it is replaced by `-b 0x100000`. Line 2 ensures that you want to update the nonvolatile memory with the alias command. Lines 3 through 6 show the output from RedBoot performing the update.



```

1 RedBoot> alias SBUF "-b 0x100000"
2 Update RedBoot non-volatile configuration - are you sure (y/n)? y
3 ... Unlock from 0x50f80000-0x50fc0000: .
4 ... Erase from 0x50f80000-0x50fc0000: .
5 ... Program from 0x0000b9e8-0x0000c9e8 at 0x50f80000: .
6 ... Lock from 0x50f80000-0x50fc0000: .

```

In this code example we see how the alias SBUF, defined earlier, is used. Line 1 shows the alias being used with the `dump`, `d` for short, command.

```

1 RedBoot> d %{SBUF}
2 0x00100000: FE03 00EA 0000 0000 0000 0000 0000 0000 |.....|
3 0x00100010: 0000 0000 0000 0000 0000 0000 0000 0000 |.....|

```

Syntax: **baudrate** [-b <rate>]

Options: -b *rate*—Baud rate to set for serial port. If this option is not specified, the current baud rate setting is displayed.

Description: Query or set the baud rate for the serial port currently in use. If the platform stores configuration information in nonvolatile memory, the baud rate is saved and used the next time the board is reset.

Syntax: **cache** [ON | OFF]

Options: ON | OFF—State to set the instruction and data caches. If this option is not specified, the current data and instruction cache settings are displayed.

Description: Query or set the state of the data and instruction caches.

Syntax: **channel** [*channel\_number*]

Options: *channel\_number*—Number of the I/O channel. Setting this parameter to -1 causes RedBoot to listen on all channels for console communication, if the `CYGPKG_REDBOOT_ANY_CONSOLE` configuration option is enabled, which is the default.

Description: Query or set the I/O channel for RedBoot to use for console communication. This command is only available for platforms with multiple I/O channels for use by RedBoot.

Syntax: **cksum** -b <location> -l <length>

Options: -b *location*—Memory address, in RAM or ROM, to begin checksum computation.  
-l *length*—Number of bytes to perform checksum calculation over.

Description: Computes the POSIX checksum on a range of memory.

Syntax: **disks**

Options: None

Description: Display disk and partition information. This command is not supported by all platforms.

Syntax: **dump** -b <location> [-l <length>] [-s] [-1 | -2 | -4]

Options: -b *location*—Address to begin memory dump.

-l *length*—Amount of data, in bytes, to dump.

-s—Dumps data in Motorola S-record format.

-1 | -2 | -4—Specifies the size of the data access. Entering -1 accesses 1 byte (8 bits) at a time where only the least significant 8 bits of the pattern are used. Entering -2

accesses 2 bytes (16 bits) at a time where only the least significant 16 bits of the pattern are used. Entering `-4` accesses 4 bytes (32 bits) at a time.

**Description:** Display a range of memory in hexadecimal format. The following shows example output from a `dump` command. Line 1 uses `du`, short for the `dump` command, with a starting address of `0x100` and a length of 32 bytes. The output is shown on lines 2 and 3.

```
1 RedBoot> du -b 0x100 -l 0x20
2 0x00000100: 3C60 0004 6044 4D4D 7C20 03A6 8080 0020 |<'...'DMM| |
3 0x00000110: 0000 0000 0000 0000 0000 0000 004B 464D |.....KFM|
```

The memory is displayed in rows of 16 bytes followed by the ASCII interpretation of the data. Care should be taken if this command is used to dump memory mapped hardware registers.

**Syntax:** **fconfig** [-i] [-l] [-d] [-n] [-f] | nickname [value]

**Options:** `-i`—Reset flash configuration settings to their default state.

`-l`—Display and edit flash configuration.

`-d`—Allows a simpler interface for the command when backspace is not allowed.

`-n`—Use nicknames for parameter entries.

`-f`—Use full names for parameter entries.

`nickname value`—Allows setting of a particular parameter entry by using the `nickname` and appropriate `value`. If `value` is not included, then the `nickname` parameter entry is displayed and a prompt for that entry is shown.

**Description:** Configure network and startup information stored in flash memory. This command is only supported on platforms that allow flash-based configuration information. As mentioned previously, the editing commands can be used with this command. Typing `return` leaves the value unchanged, period can be used to halt editing of parameters, and caret can be used as an up arrow to go back to editing a previous parameter. After entering the flash configuration settings, a prompt is displayed asking whether to write the new configuration into flash memory.

The following shows output from the `fconfig` command. Line 1 shows the command with the option to list the current configuration. Lines 2 through 7 show the output for the current configuration on this particular platform.

```
1 RedBoot> fconfig -l
2 Run script at boot: false
3 Use BOOTP for network configuration: false
4 Local IP address: 192.168.1.29
5 Default server IP address: 192.168.1.101
6 GDB connection port: 9000
7 Network debug at boot time: false
```

This information is only used at reset; therefore, in order for changes to take effect, the platform must be restarted. A scripting example is shown in the *Boot Scripting* section of this chapter.

**Syntax:** **fis** [command]

**Options:** `command`—The different FIS commands are detailed in Item List 9.3.

**Description:** Flash Image System (FIS) allows RedBoot to use flash memory for executable and data image storage. FIS commands are available on platforms that support flash memory. There is a subset of commands associated with the FIS that are covered in Item List 9.3.

- Syntax: **go** [-w <timeout>] [entry]  
 Options: -w timeout—Amount of time, in seconds, to wait before beginning program execution. Allows aborting of program execution by typing CTRL-C on the console. If this option is not specified, program execution begins immediately.  
 entry—Location in memory to begin program execution. If this parameter is not specified, the entry point of the last loaded image is used.  
 Description: Execute a program.
- Syntax: **help** [<topic>]  
 Options: topic—Command to display help information about.  
 Description: Display information about the RedBoot command set.
- Syntax: **ip\_address** [-l local\_IP\_addr] [-h server\_IP\_addr] [-d dns\_server\_IP\_addr]  
 Options: -l local\_IP\_addr—The IP address for RedBoot.  
 -h server\_IP\_addr—The IP address of the default server. This address is used by other commands, such as load.  
 -d dns\_server\_IP\_addr—The IP address of the DNS server.  
 Description: Query or set the RedBoot, server, and DNS server IP address.
- Syntax: **load** [-r] [-v] [-d] [-h <host>] [-m [[TFTP] | [HTTP] | [xyMODEM] | [disk]] -c <channel\_number>] [-b <base\_address>] <file\_name>  
 Options: -r—Download raw data into memory. Using this option also requires the -b option to be specified.  
 -v—Display an indicator while the download is in progress. This is useful for feedback during long downloads.

---

**NOTE** The -v option should not be used when downloading over the network as it can slow the loading process.

- d—Decompress gzipped image during download.
- h host—Used in TFTP and HTTP modes only to specify the host computer for the download. The host parameter can be specified as an IP address or a hostname if DNS is enabled.
- m TFTP|HTTP|xyMODEM|disk -c channel\_number—Specifies the download method to use. The choices are TFTP or HTTP for network-based downloads, xMODEM or yMODEM for serial-based downloads, and disk for downloads from hard disk or CD-ROM (which is not supported on all platforms). The -c option allows specification of the channel used for the download. This option is not available on all platforms, since some platforms only contain one I/O channel for RedBoot console communication.
- b base\_address—Address location in memory to load the file. Typically, executable images are loaded into the memory location to which the file was linked; however, this option allows the overriding of the image's linked location.
- file\_name—Filename to download. This parameter is not required for the serial-based xyMODEM downloads.

**Description:** Download data to the target system. Data can be loaded over the serial port using the X or Y modem protocols, over the network port using the TFTP protocol, over the network using the HTTP protocol, or from a storage disk (the disk method is not supported by all platforms). There are several free TFTP servers available on the Web, and most terminal programs, included with the host operating system, support the different modem protocols.

**Syntax:** **mcmp** -s mem\_addr1 -d mem\_addr2 -l length [-1 | -2 | -4 ]

**Options:**  
 -s mem\_addr1—Memory start address of first location for data comparison.  
 -d mem\_addr2—Memory start address of second location for data comparison.  
 -l length—Length of data.  
 -1 | -2 | -4—Specifies the size of data access for comparison. Entering -1 accesses 1 byte (8 bits) at a time where only the least significant 8 bits of the pattern are used. Entering -2 accesses 2 bytes (16 bits) at a time where only the least significant 16 bits of the pattern are used. Entering -4 accesses 4 bytes (32 bits) at a time.

**Description:** Compare the contents of two memory address ranges. If the buffers do not match, only the first nonmatching element is displayed.

**Syntax:** **mfill** -b addr -l length -p value [-1 | -2 | -4 ]

**Options:**  
 -b addr—Memory start address for data fill.  
 -l length—Length of data.  
 -p value—Value to fill into memory.  
 -1 | -2 | -4—Specifies the size of data access for comparison. Entering -1 accesses 1 byte (8 bits) at a time where only the least significant 8 bits of the pattern are used. Entering -2 accesses 2 bytes (16 bits) at a time where only the least significant 16 bits of the pattern are used. Entering -4 accesses 4 bytes (32 bits) at a time.

**Description:** Fill a memory range with the specified data.

**Syntax:** **ping** [-v] [-n <count>] [-l <length>] [-t <timeout>]

**Options:**  
 [-r <rate>] [-i <IP\_address>] [-h <IP\_address>  
 -v—Verbose setting that allows the display of information about each packet sent.  
 -n count—Number of packets to send. If this option is not specified, 10 packets are sent by default.  
 -l length—Size of payload for ping packet. The default size of a ping packet when this option is not specified is 64 bytes. The maximum value for this option is 1400.  
 -t timeout—Length of time in milliseconds (ms) to wait for the host to return a sent packet. The default timeout is 1000 ms if this option is not specified.  
 -r rate—Time in milliseconds (ms) between sending ping packets. The default is 1000 ms. If 0 is specified for the rate, packets will be sent without delay.  
 -i IP\_address—Allows overriding of IP address in outgoing ping packets.  
 -h IP\_address—Specifies the IP address, or a hostname can be used if DNS is enabled, of the host to send the ping packets.

**Description:** Checks the connectivity of the local network by sending ICMP ping packets to the specified host. The host returns the ping packets, if received, and data for each packet sent and received is displayed.

**Syntax:** **reset**

**Options:** None

Description: Reset the target system. This is equivalent to a power-on reset. Not all platforms support this command.

Syntax: **version**

Options: None

Description: Display the RedBoot version information. The output from the `version` command is shown in the following listing.

```

1 RedBoot> version
2 RedBoot(tm) bootstrap and debug environment [FLOPPY]
3 Non-certified release, version UNKNOWN - built 17:34:58, Jun 10 2002
4 Platform: PC (I386)
5 Copyright (C) 2000, 2001, 2002, Red Hat, Inc.
6 RAM: 0x00000000-0x000a0000, 0x00088870-0x000a0000 available

```

In the preceding listing, line 1 shows the command entry. Line 2 displays the RedBoot title message and Startup Type for which the image was built. The build information is displayed on line 3. Line 4 displays the platform information. Copyright information is output on line 5. Finally, line 6 shows the total RAM resources and what is available for use. This message is similar to the initialization message shown in Code Listing 9.1.

Syntax: **x** `-b <location> [-l <length>] [-s] [-1 | -2 | -4]`

Options: `-b location`—Address to begin memory dump.

`-l length`—Amount of data, in bytes, to dump.

`-s`—Dumps data in Motorola S-record format.

`-1 | -2 | -4`—Specifies the size of the data access. Entering `-1` accesses 1 byte (8 bits) at a time where only the least significant 8 bits of the pattern are used. Entering `-2` accesses 2 bytes (16 bits) at a time where only the least significant 16 bits of the pattern are used. Entering `-4` accesses 4 bytes (32 bits) at a time.

Description: Alias for `dump` command.

---

#### NOTE A Little Trick...

The GDB stub in the RedBoot image can be used to load an image into RAM as well. This causes you to lose control of the RedBoot prompt. However, entering the command `++$k#6b` directly at the serial port causes RedBoot to restart, retaining the image loaded into RAM.

RedBoot can use flash memory for data and executable image storage. Item List 9.3 details the FIS command set.

#### Item List 9.3 Flash Image System Commands

Syntax: **fis create** `-b <mem_base> -l <length> [-f <flash_addr>] [-e <entry_point>] [-r <ram_addr>] [-s <data_length>] [-n] <name>`

Options: `-b mem_base`—Location in RAM of data to be stored in flash memory.

`-l length`—Length of the data in RAM.

`-f flash_addr`—Location in flash memory for the image. The flash address can be included as part of the linked executable image. If this option is not specified, the first free block in flash memory is used.

`-e entry_point`—Execution entry address.

`-r ram_addr`—Location in RAM when the image is loaded using the `fis load` command. This option only needs to be specified if the image will be loaded with the `fis load` command.

`-s data_length`—Length in bytes of the actual data to be written to flash memory. If this option is not specified, the image length of the `-l` option is used. If `data_length` is less than the `length` for the `-l` option, the remainder of the image in flash memory is left blank.

`-n`—Updates FIS directory without copying image data from RAM to flash memory. This option can be used to recreate the FIS entry if it has been destroyed.

`name`—Name of the image to create.

Description: Create an image in the FIS directory, which causes data currently in RAM to be stored in flash memory. Data for the image must be present in RAM memory prior to calling this command.

Syntax: **fis delete** name

Options: name—Name of the image to delete.

Description: Remove the image, specified by the `name` parameter, from the FIS, which causes the image to be erased from flash memory and removed from the FIS directory.

Syntax: **fis erase** `-f` <flash\_addr> `-l` <length>

Options: `-f flash_addr`—Start flash memory address to erase.

`-l length`—Length in bytes to erase.

Description: Force an erase of a portion of flash memory. No checking is done to ensure whether the area specified corresponds to a loaded image.

Syntax: **fis free**

Options: None

Description: Display areas of flash memory that are currently not in use, meaning that the block contains non-erased contents. This command can be to check if a particular location is in use prior to loading an image.

Syntax: **fis init** [`-f`]

Options: `-f`—Erase or format all blocks of flash memory.

Description: Initialize the FIS. This command should only be executed once during the first installation of RedBoot on the target hardware. Executing this command more than once causes the loss of data in flash memory.

Syntax: **fis load** [`-b` <memory\_load\_addr>] [`-c`] [`-d`] name

Options: `-b memory_load_addr`—Location in RAM to copy the image from flash memory. If this option is not specified, the image is copied from flash memory to the address given when the image was created. Executable images normally load at their linked location.

`-c`—Compute and display the checksum of the image data after loading is complete.

- d—Decompress gzipped image while copying the image to RAM.  
name—Name of the image.
- Description: Transfer an image from RAM into flash memory.
- Syntax: **fis list** [-c] [-d]
- Options: -c—Display image checksum instead of the memory address field.  
-d—Display image data length instead of amount of flash used.
- Description: Display images currently stored in the FIS.
- Syntax: **fis lock** -f <flash\_addr> -l <length>
- Options: -f flash\_addr—Start flash memory address to lock.  
-l length—Length in bytes to lock.
- Description: Write-protect a portion of flash memory, preventing overwriting of stored images. This command is only available on platforms that support write-protection of flash memory.
- Syntax: **fis unlock** -f <flash\_addr> -l <length>
- Options: -f flash\_addr—Start flash memory address to unlock.  
-l length—Length in bytes to unlock.
- Description: Unlock a portion of flash memory, allowing updating of that portion's contents. This command must be used for portions of flash memory previously locked before the FIS can use those areas of memory.
- Syntax: **fis write** -b <location> -l <length> -f <flash\_addr>
- Options: -b location—Start RAM address to begin reading data.  
-l length—Length of bytes to write.  
-f flash\_addr—Start flash memory address to write data.
- Description: Write data from RAM to flash memory.

As mentioned previously, the RedBoot command set offers basic functionality for debugging and control over target hardware. The RedBoot source code is available if you need to extend the command set or functionality offered by the current commands. After adding the necessary features to RedBoot, a new image needs to be created. The steps involved in creating a RedBoot image are covered in Chapter 11. Next, the current RedBoot image on your target hardware needs to be updated. Updating the RedBoot image is covered in Chapter 12.

#### 9.4.1.1 Boot Scripting

RedBoot supports running scripts during the target hardware boot cycle. Boot scripting allows you to run commands you always want executed after power up. For example, if you are constantly going to load an image into memory for execution and debug, you can avoid entering commands manually by setting up a boot script. A timeout is included in the boot script setup to allow you to abort running the boot script during a particular power up cycle by simply pressing *Ctrl+C*. This boot script is set up and run on the Motorola PowerPC MBX860 board.

Boot scripts are set up using the `fconfig` command. Entering `true` (or `t`) when `fconfig` prompts for `Run script at boot` puts RedBoot in a mode allowing you to

begin inputting your boot script commands. The setup for a simple boot script is shown in Code Listing 9.2.

```

1 RedBoot> fconfig
2 Run script at boot: false t
3 Boot script:
4 Enter script, terminate with empty line
5 >> fi li
6 >>
7 Boot script timeout: 0 10
8 Use BOOTP for network configuration: false .
9 Update RedBoot non-volatile configuration - are you sure (y/n)? y
10 ... Erase from 0xfe060000-0xfe070000: .
11 ... Program from 0x0000f810-0x0000fc10 at 0xfe060000: .
12 RedBoot>

```

**Code Listing 9.2** RedBoot script setup example.

The bold characters in Code Listing 9.2 are values entered from the keyboard. Line 1 shows the command entry. Line 2 shows selecting `t` to enter boot script mode; the `false` shown on this line is the current value. The actual command to run during the boot script execution is shown on line 5. This is the `fi` `li` command, or `fi li` for short. Line 6 contains an empty line input to terminate the boot script. Line 7 shows that a timeout of 10 seconds is entered; the default is 0. On line 8, a period (`.`) is entered to terminate any more entries for the `fconfig` command. Lines 9 through 11 show the update of the configuration information in nonvolatile memory.

```

1 FLASH: 0xfe000000-0xfe080000,8 blocks of 0x00010000 bytes each.
2 IP: 192.168.0.100, Default server: 192.168.0.3, DNS server IP: 0.0.0.0
3
4 RedBoot(tm) bootstrap and debug environment [ROM]
5 Non-certified release, version UNKNOWN - built 04:02:12, Jul 06 2002
6
7 Platform: Motorola MBX (PowerPC 860)
8 Copyright (C) 2000, 2001, 2002 Red Hat, Inc.
9
10 RAM:0x00000000-0x00400000, 0x00022420-0x003e0000 available
11 == Executing boot script in 10 seconds - enter ^C to abort
12 RedBoot> fi li
13 Name FLASH addr Mem addr Length Entry point
14 RedBoot 0xFE000000 0xFE000000 0x00020000 0x00000000
15 RedBoot[backup] 0xFE020000 0xFE020000 0x00020000 0x00000000
16 RedBoot config 0xFE060000 0xFE060000 0x00010000 0x00000000
17 FIS directory 0xFE070000 0xFE070000 0x00010000 0x00000000
18 RedBoot>

```

**Code Listing 9.3** RedBoot script example output.



Code Listing 9.3 shows the output from the simple boot script we set up in Code Listing 9.2. The power to the board was cycled in order to get RedBoot to restart. The RedBoot initialization message is shown on lines 1 through 10.

Line 11 is the message showing that the boot script is going to execute. At this point, *Ctrl+C* can be entered to abort running the boot script. On line 12 we see the `fi li` boot script that we set up in Code Listing 9.2 execute. We can see the output from the `fi s list` command on lines 13 through 17. Finally, the script ends and RedBoot is ready for additional commands, as shown on line 18.

## 9.5 Summary

In this chapter, we covered the RedBoot ROM monitor. Using RedBoot, we are able to load and debug applications. We looked at the various configuration options for RedBoot, as well as the different communication mechanisms available. Finally, we went into details about using RedBoot by exploring the user interface and command set available.

# The Host Development Platform

**T**his is the first of four chapters that covers the configuration of the host development platform, the eCos toolset, examples using the eCos system, and porting eCos. These chapters guide us through the steps necessary to configure the different components of the eCos development system. At the end of these chapters, you will have a complete embedded software development environment and an understanding of how to build an application using the eCos system.

To start, in this chapter, we get an overview of the main steps involved with configuring our host PC, focusing on setting up the tools needed to build eCos.

Next, in Chapter 11, *The eCos Toolset*, we proceed with the installation of the eCos configuration tools and the source code repository. This lays the foundation for building the eCos RTOS library.

In Chapter 12, *An Example Application Using eCos*, allows us to put our tools to work by developing an application to run on our development system. We focus on the target-specific compiler and debugging tools. This chapter offers a basic example in which additional functionality and components can be added to develop a fully featured, embedded application.

Finally, Chapter 13, *Porting eCos*, we look at the steps necessary to move eCos onto a new hardware platform. This chapter lays the foundation for getting eCos up and running on your own product.

## 10.1 Overview

The eCos development system is available on the Internet. In this chapter, we focus on setting up the eCos development system using the Net release since it is freely (no up-front costs to get started) available to everyone—provided that you abide by the license terms for each tool or program.

The CD-ROM accompanying this book contains all of the files needed to set up the eCos development environment described in this book. Links are given to show exactly where the files can be obtained online. Source code is provided for the executable tools contained on the CD-ROM. This is one method for satisfying the requirements of the GNU General Public License (GPL). The GPL (version 2) is contained in the latter part of Appendix B, *eCos License*, and can be found online at:

[www.gnu.org/licenses/gpl.html](http://www.gnu.org/licenses/gpl.html)

The files on the CD-ROM have actually been used to set up the eCos development suite described in this book. Since these are open-source tools, they are constantly evolving and being updated.

The readme file, `Readme.txt`, included on the CD-ROM under the root directory contains additional information about the contents of the CD-ROM.

---

**NOTE** You might find that you need to use a newer version of a particular tool that incorporates specific features or bug fixes. Information on upgrading the Cygwin tools is provided in Appendix C, *Cygwin Tools Upgrade Procedure*.

However, the general rule to follow concerning the tools is:

*If it ain't broke, don't fix it.*

Basically, if your installed toolset can accomplish all of your development tasks, it is best to leave the system in its current working state. Leave the upgrade for later when you have time to devote to it. Alternatively, configure and use a system other than your main development environment platform.

The best approach for managing your toolset is to get a stable working development platform using the tools provided on the CD-ROM. Then, prior to performing any upgrade, ensure that you leave yourself a backdoor (by backing up any necessary files) to restore your current working environment before proceeding with any upgrades. This will guarantee that you always have a functioning development environment to use.

The eCos development suite can be set up to run under two operating systems—Linux and Windows. The Linux configuration has been tested on Red Hat Linux version 7.0 (or later) for x86.

The Windows configuration has been tested using Windows NT 4.0, although Windows 2000, Windows XP, Windows 98, and Windows 95 can also be used.

In this book, we use the Windows NT 4.0 (Service Pack 6.0a) operating system for our host development platform. You can find information on configuring the Linux operating system for eCos development on the eCos home page.

## 10.2 Configuring the Windows Host

In order to develop eCos applications using the Windows operating system, tools must be installed to build the eCos cross compilers and other development tools. The tools that run under the Windows operating system are called *Cygwin*. Cygwin, an acronym for GNU Cygnus Windows, is a UNIX environment for Windows consisting of a Dynamic Link Library (DLL) that acts as a UNIX emulation layer and a collection of UNIX tools ported to Windows. The Cygwin site is located at:

<http://cygwin.com>

The Cygwin tools are capable of running on Windows 95/98/ME/NT/2000/XP operating systems, but not on Windows CE. The basic steps involved in setting up the eCos development system on a Windows NT host are:

- 1. Install Cygwin, the GNU native development tools for Windows**—The native host tools allow us to build the eCos toolset. This information is covered in this chapter.
- 2. Build the platform-specific cross-development tools**—The Cygwin tools allow us to build the development tools we need to construct the eCos RTOS and applications for a specific processor. We cover this process for the x86 specific tools in this chapter. The details for building the tools for other processors are also described in this chapter.
- 3. Install the eCos development kit**—The eCos development kit contains a release of the eCos source code repository and configuration tools for building the eCos kernel. The development kit installation is covered in this chapter.
- 4. Set up and configure the eCos Configuration Tool**—The Configuration Tool is used, among other things, to assist in creating the application-specific eCos library files. This final procedure is detailed in Chapter 11.

---

**NOTE** It is a good idea when using the Windows operating system for your host development computer to use the NTFS or FAT32 file systems on the drive on which the tools are installed. The tools contain numerous small files, which can waste more disk space if the hard drive is not formatted with NTFS or FAT32.

Table 10.1 lists the disk space used by each of the different host development modules for the Windows platform.

We are going to be building the x86 platform cross-development tools. This allows us to develop and debug applications using a second PC as our target hardware. We cover examples using the target PC hardware in Chapter 12.

*And away we go...*

**Table 10.1** Windows Host Disk Space Requirements

| Module                      | Disk Space Required       |
|-----------------------------|---------------------------|
| Cygwin                      | 374.2 Mbytes <sup>a</sup> |
| x86 Cross-Development Tools | 74.1 Mbytes               |
| eCos Development Kit        | 81.8 Mbytes               |
| <b>TOTAL</b>                | <b>530.1 Mbytes</b>       |

<sup>a</sup> Includes source code. Without source code, the Cygwin tools require 202.2 Mbytes of disk space.

### 10.2.1 Installing the Cygwin Native Tools

The Cygwin files needed to install the tools are included on the CD-ROM under the `cygwin` directory. This directory includes the setup program, `setup.exe`, and the subdirectories `contrib` and `latest`, which contain the Cygwin executable tools and the source code files as well. The tool files have names such as `cygwin-1.3.3-2.tar.bz2`, while the source code files contain `src` in the filename; for example, `cygwin-1.3.3-2-src.tar.bz2`. All of the files necessary for installing the Cygwin tools can also be found online at:

<http://cygwin.com/download.html>

If traffic is busy on the main Cygwin site, it might be necessary to use one of the mirror sites. A list of Cygwin mirror sites can be found online at:

<http://cygwin.com/mirrors.html>

The Cygwin readme file, `readme.txt`, is also contained on the CD-ROM in the `cygwin` directory. It is a good idea to read this file prior to beginning the Cygwin tools installation. The readme file can also be found online at any of the Cygwin distribution sites. For any additional information or specific problems installing or using the Cygwin tools, you can look at the Cygwin discussion mailing list located online at:

<http://cygwin.com/ml/cygwin>

Posts to this discussion list should be sent to `cygwin@cygwin.com`. Additional discussion Cygwin discussion mailing lists can be found online at:

<http://cygwin.com/lists.html>

Manuals for the different Cygwin development tools can be found online at:

[www.gnu.org/manual](http://www.gnu.org/manual)

---

**NOTE** Before proceeding with the installation it is important to ensure that all Cygwin applications, if any, running on your system are shut down. If you have never installed any Cygwin tools on your system, you do not have to worry about this.

It is also a good idea to disable any anti-virus software you have running. Occasionally conflicts occur, such as system hangs, between the Cygwin files and certain anti-virus software.

Having a space in your Windows logon name might also cause a problem using the Cygwin tools. To avoid any problems, it is best to install using a Windows logon name without spaces.

If you are upgrading a previous version of the Cygwin tools, you should refer to the `readme.txt` file in the `cygwin` directory on the CD-ROM to guarantee that the proper steps are performed prior to installation.

### STEP 1

The first step for installing Cygwin is to create a directory for the tools on the hard drive. It is preferable to create this directory off one of the root drives in order to keep path names short. We are going to use the directory name `cygwin` off the root `D:\` drive.

### STEP 2

Next, we copy the contents of the `cygwin` directory from the CD-ROM to the directory we just created. This takes up approximately 178.4 Mbytes of disk drive space, which includes the source code for the tools.

---

**NOTE** The Cygwin setup program can be pointed to the `cygwin` directory on the CD-ROM (`E:\cygwin`, where `E:` is your CD-ROM drive) in STEP 5 for installation to eliminate copying the files on the hard drive. However, having the files on the hard drive can be helpful when you decide to upgrade the Cygwin tools.

### STEP 3

Run `setup.exe` from the directory we just created on the hard disk. Next, we go through the procedure for configuring the setup program to install the proper Cygwin packages. Illustrations of the dialog boxes and selection of the options for this installation process are provided.

To get started, we run `setup.exe` from the `D:\cygwin` directory.

The first dialog box after running the Cygwin setup is shown in Figure 10.1. This dialog shows us the version information for the setup program. We are using setup version `2.78.2.15`. It is important to note the version in case you need to get help from a discussion list or you need to upgrade to a newer version in the future.

Click the Next button to proceed with installation. The latest version of the `setup.exe` program can be found online at:

<http://sources.redhat.com/cygwin/setup.exe>



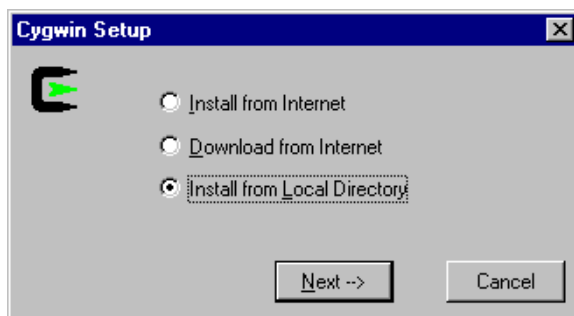
**Figure 10.1** Cygwin setup version dialog box.

Click the Next button to proceed with installation.

#### STEP 4

The next step in the Cygwin installation is to select the location we want to install from. The options are shown in the dialog box in Figure 10.2. We select *Install from Local Directory* and click the Next button.

We can use the *Install from Internet* option, as shown in the *Upgrading the Cygwin Tools* section of this chapter, if we need to upgrade the Cygwin tools in order to download and install the latest versions of the tools available online. The *Download from Internet* option allows us to download the latest version of the tools, both executable and source code, without performing the install.



**Figure 10.2** Cygwin setup file location dialog box.

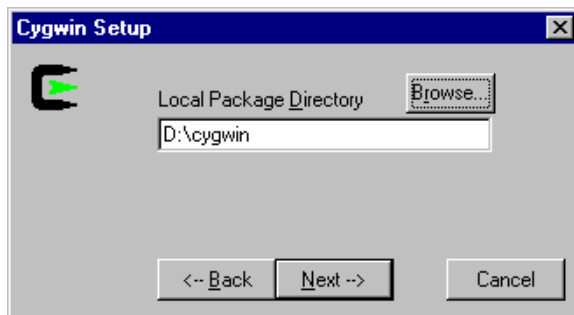
#### STEP 5

Now we want to select the location of the Cygwin packages we want to install, *Local Package Directory*. The dialog box for this option is shown in Figure 10.3. We set this *Local Package Directory*

option to `D:\cygwin` by either typing it in directly or clicking the Browse button to find the proper directory. Then, click Next.

**NOTE** If you did not copy the Cygwin tools to the hard drive, enter `E:\cygwin` to install from the CD-ROM drive, where `E:` is your CD-ROM drive letter.

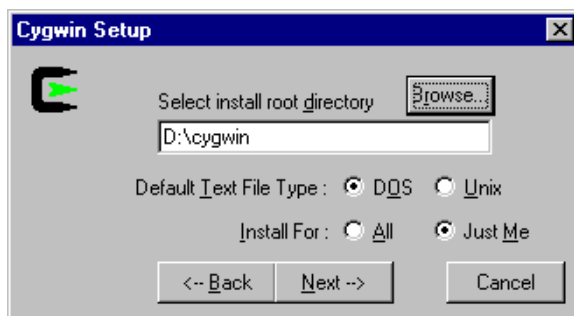
As we can see in Figure 10.3, the setup allows us to go back to a previous dialog box by using the Back button.



**Figure 10.3** Cygwin local package directory selection dialog box.

## STEP 6

Next, we set up the location where we want the tools installed, *Select Install Root Directory*. This dialog box is shown in Figure 10.4. We set this option to `D:\cygwin` by either typing it in directly or clicking the Browse button to find the proper directory.



**Figure 10.4** Cygwin install root directory selection dialog box.

We also need to select the *Default Text File Type* and *Install For* options. The *Default Text File Type* allows us to choose *DOS*, in which case text files will end with `\r\n`, or *Unix*, meaning text files will end with `\n`. Select *DOS* for this option.



The *Install For* option allows us to select *All*, which allows anyone that logs on to the host PC to have access to the Cygwin mount table, or *Just Me*, if you are the only one needing access. Select *Just Me* for this option. We can now click the Next button.

### STEP 7


The next step is to select the packages we want to install. The packages contained on the CD-ROM might not be the latest versions available because changes to the Cygwin tools are continuously occurring. However, the CD-ROM files have been installed and configured into a working eCos development system.

The CD-ROM contains the Cygwin DLL version 1.3.3-2. We are using version 1.3.3-2 because it has been verified to work with the eCos development tools. If you look back at some of the posts on the eCos discussion list, you can see the issues that have come up using new versions of the Cygwin DLL. To avoid this, it is best to get a known working development environment up and running before upgrading to any new versions. Then, if problems occur, you can always restore your previous working environment.

The dialog box for selecting specific Cygwin packages is shown in Figure 10.5. The radio buttons at the top—*Prev*, *Curr*, and *Exp*—allow us to select the previous, current, or experimental versions, respectively, for each Cygwin package. We want to ensure that *Curr* is selected, which is the default option. Selection of individual versions for a specific package is described later.

The *Full/Part* button allows us to display the full list of packages or only a partial list. This is typically used for package upgrades.

The columns in the dialog box display information about the package. The first column, *Current*, shows the current release for a particular package. This column is used for package upgrades. Since this is a new install, all package version details are contained in the column *New*. The *New* column shows the version that is going to be installed. The *Src?* column allows us to select whether we want source code installed for the particular package. The last column, *Package*, shows the name of the package.

As described previously, the *Prev*, *Curr*, and *Exp* radio buttons allow us to select particular versions for all packages. In order to select specific versions of a single package, we must click on the  icon in the *New* column. This toggles the action or version for a specific package to one of the following:

- **Skip**—skip installation of the package.
- **Source**—install source code only for the package.
- **Uninstall**—remove the package.
- **Keep**—leaves the current version of the package.
- **Version**—install the selected version for the package.

For certain actions, the *Src?* column is not relevant, in which case n/a is displayed in this column. The *cygwin* package is highlighted in gray in Figure 10.5 to help shown the proper version of the package we want to use for our installation.

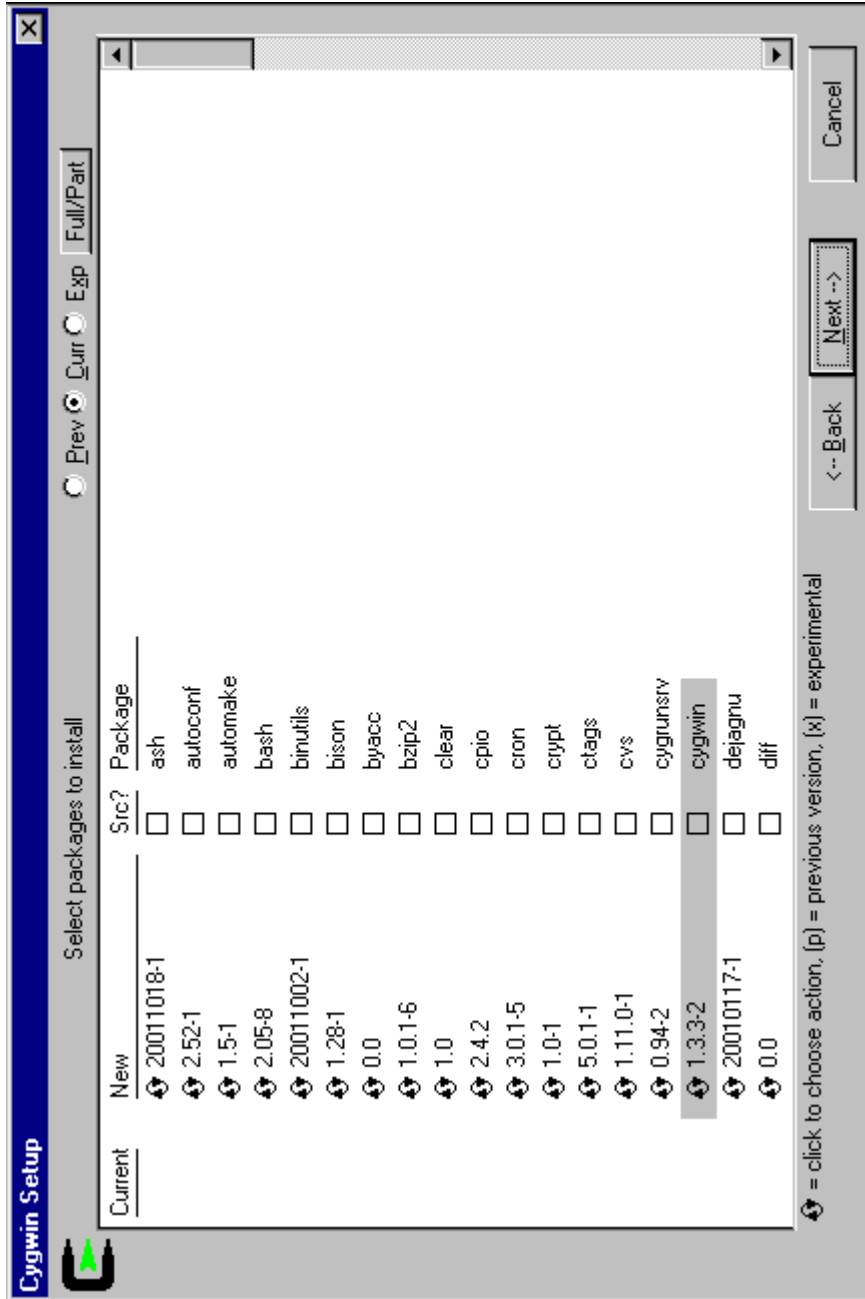
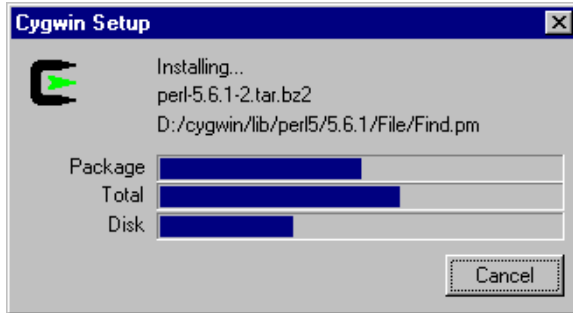


Figure 10.5 Cygwin package selection dialog box.

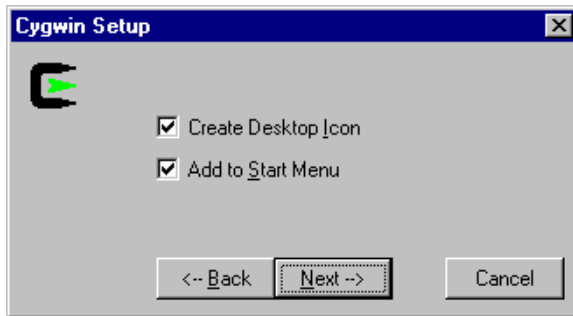
We are now ready to begin the installation, which happens by clicking the Next button. As installation progresses, a dialog box similar to the one shown in Figure 10.6 is displayed. This shows the current package being installed, the location for installation, and progress indicators for the package and total installation procedure.



**Figure 10.6** Cygwin package installation progress dialog box.

### STEP 8

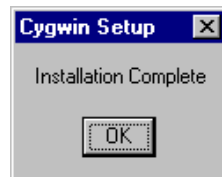
The final step in the Cygwin tools installation is to select whether desktop and Start menu shortcuts should be added to run the Cygwin environment shell program. The dialog box for these options is shown in Figure 10.7. Clicking Next creates the shortcuts according to the options selected.



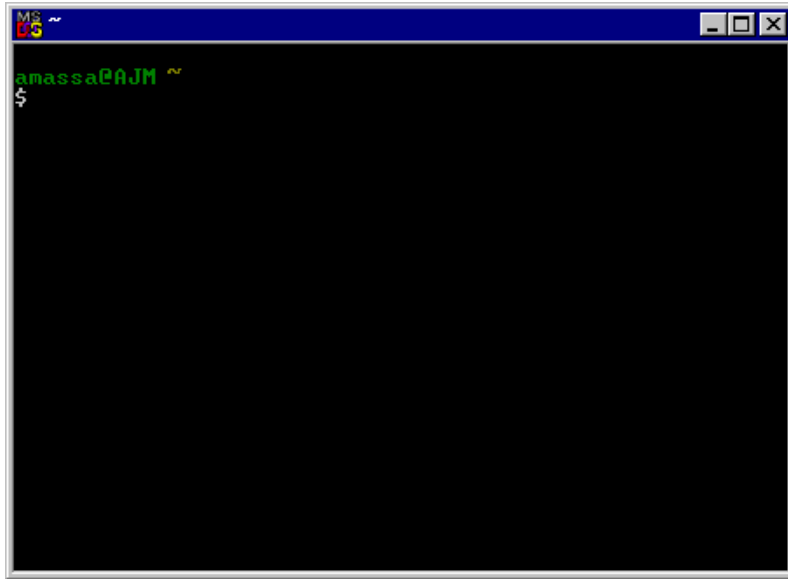
**Figure 10.7** Cygwin shortcut setup dialog box.

After successful installation of the Cygwin tools, the dialog box shown in Figure 10.8 is displayed. Clicking OK completes the Cygwin tools installation.

**Figure 10.8** Cygwin tools installation complete dialog box.



To ensure proper installation of the Cygwin tools, we can run the bash program by double-clicking on the desktop shortcut created in . This brings up a UNIX bash shell environment similar to the one shown in Figure 10.9. To close the bash shell, type `exit` (or the shortcut Ctrl+D) at the bash shell prompt (`$`).



**Figure 10.9** Cygwin bash shell program.

## STEP 9

We now need to add the `cygwin\bin` directory to the Windows environment path. The path is altered by right-clicking on the My Computer icon on the desktop. This brings up a drop-down list of options. Select Properties from the drop-down list.

The System Properties dialog box is displayed. Select the Environment tab. Under the User Variables, select path. In the Value edit box, to the front of the path, add:

```
D:\cygwin\bin;
```

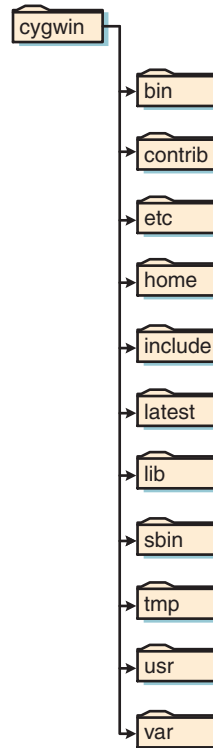
Then, click the Set button. Finally, click the OK button.

### 10.2.1.1 Cygwin Tools Directory Structure

Now that we have successfully completed the installation of the Cygwin tools, we can take a look at the directory structure created, which is shown in Figure 10.10. If disk space is a concern the `contrib` and `latest` subdirectories can be deleted, since these are on the CD-ROM anyway.

We can now take a quick look at some of the subdirectories in the Cygwin tools directory structure. The root `cygwin` directory includes the file `cygwin.bat`, which is the batch file

**Figure 10.10** Cygwin installation directory structure.



that is run when clicking on the *Cygwin* shortcut created on the desktop or clicking on *Cygwin Bash Shell* under *Start -> Programs -> Cygnus Solutions* (provided these were created in Step 8 of the Cygwin tools installation). The bash program is the UNIX environment shell that we use to create the eCos development tools, which is described later in this chapter.

One note: After running the bash program for the first time, the `home` subdirectory is created under the `cygwin` root. The `home\xxx`, where `xxx` is your username, subdirectory includes a file that contains the history of commands run in the bash shell environment.

The `bin` subdirectory contains all of the executable files for the Cygwin tools. Included in this subdirectory is the Cygwin DLL, `cygwin1.dll`, which we are using version 1.3.3. The `etc` subdirectory contains standard UNIX configuration files. The `lib` subdirectory contains all of the library files used when developing under Cygwin. The `tmp` subdirectory is the temporary scratch directory for the Cygwin tools. Finally, the `usr` subdirectory contains the miscellaneous files stored by the different Cygwin packages.

The root directory also contains two log files, `setup.log` and `setup.log.full`. An example of some of the information contained in the `setup.log` file is shown in Code Listing 10.1. Line 1 gives the starting date and time of the installation and the version (2.78.2.15) of the `setup.exe` file used. Lines 2 through 5 contain the option information, such as the location of the installation files, we selected during the Cygwin tools installation.

Information about the installation for each package is shown on lines 6 through 21. These lines display the subdirectory, either `contrib` or `latest`, of the package for installation. Lines 22, 23, and 24 simply show the place that the `setup.log` file information was cut out. Lines 25 and 26 are the last two lines of the file that show the completion message and the end date and time of the install. We can see comparing the start and end message that the complete Cygwin tools install took approximately 16 minutes.

The file `setup.log.full` is similar to `setup.log`; however, additional detailed information about the setup for each package is contained in the full log file. Any time the `setup.exe` program is run, the operations performed are logged into these two log files. This is helpful to keep track of package upgrades for the different Cygwin tools.

```
1 2001/09/27 12:12:12 Starting cygwin install, version 2.78.2.15
2 Current Directory: D:\cygwin
3 source: from cwd
4 Selected local directory: D:\cygwin
5 root: D:\cygwin text user
6 Installing test version...latest/ash/ash-20011018-1.tar.bz2
7 Installing...latest/autoconf/autoconf-2.52-1.tar.bz2
8 Installing...latest/automake/automake-1.5-1.tar.bz2
9 Installing...latest/bash/bash-2.05-8.tar.bz2
10 Installing test version...latest/binutils/binutils-20011002-1.tar.bz2
11 Installing...latest/bison/bison-1.28-1.tar.gz
12 Installing...latest/byacc/byacc.tar.gz
13 Installing...latest/bzip2/bzip2-1.0.1-6.tar.gz
14 Installing...latest/clear/clear-1.0.tar.gz
15 Installing...latest/cpio/cpio-2.4.2.tar.gz
16 Installing...contrib/cron/cron-3.0.1-5.tar.bz2
17 Installing...latest/crypt/crypt-1.0-1.tar.gz
18 Installing...latest/ctags/ctags-5.0.1-1.tar.gz
19 Installing...contrib/cvs/cvs-1.11.0-1.tar.gz
20 Installing...latest/cygrunsrv/cygrunsrv-0.94-2.tar.bz2
21 Installing previous version...latest/cygwin/cygwin-1.3.3-2.tar.bz2
22 .
23 .
24 .
25 mbox note: Installation Complete
26 2001/09/27 12:18:34 Ending cygwin install
```

**Code Listing 10.1** Cygwin installation example log file contents.

### 10.2.1.2 Upgrading the Cygwin Tools

The Cygwin tools upgrade procedure is detailed in Appendix C, in case you need bug fixes or enhancements offered in the latest versions of the Cygwin tools.

---

**NOTE** Do not go through the Cygwin tools upgrade procedure on your development platform at this point. If you do, your platform will have different Cygwin tools than those tested in this book. This section is for future reference when you need, or want, to upgrade the Cygwin tools.

### 10.2.2 Installing the Platform-Specific Cross-Development Tools

Next, we install the tools that allow us to build eCos and our applications for our specific processor; in our case, the Intel x86. The cross-development tools are used to build the eCos library for our specific platform; in our case, the Intel x86. We also use these same tools when building applications, as we see in Chapter 12.

Rather than go through the procedure of configuring and building the i386 GNU cross-development tools, pre-built versions are supplied on the CD-ROM. This eliminates the possibility of generating incorrect GNU tools, which is very easy considering the command strings that need to be entered. These tools are used in the examples included with this book. The procedure used to configure and build the GNU cross-development tools is contained in Appendix D, *Building the GNU Cross-Development Tools*.

---

**NOTE** Pre-built binary versions of the PowerPC GNU cross-development tools are also included on the CD-ROM in the file `ppcgnutools.tar.bz2` under the `gnu\ppctools` directory. These tools are used in Chapter 13.

The platform-specific cross-development tools can be broken down into three different groups:

- GNU Binary Utilities (commonly called binutils)
- GNU C/C++ Compiler
- GNU Insight Debugger with Insight Interface

The file needed to install the GNU cross-development tools is located on the CD-ROM under the `gnu` subdirectory and is called `i386gnutools.tar.bz2`. The versions we are using of the GNU cross-development tools are shown in Table 10.2.

**Table 10.2** GNU Cross-Development Tools Versions

| Tool                                | Version |
|-------------------------------------|---------|
| GNU Binary Utilities                | 2.11.2  |
| GNU C/C++ Compiler                  | 2.95.2  |
| GNU Debugger with Insight Interface | 5.1     |

Although the GNU Binary Utilities and GNU C/C++ Compiler versions are not the latest available, they have been confirmed to work in the development environment used in this book for the i386 PC target platform. It might be necessary for you to upgrade to newer tool versions if different processor architectures are used. Prior to upgrading, it is a good idea to search the eCos discussion list to see if there have been any posts describing problems with a newer version.

---

**NOTE** The GNU C/C++ Compiler version 2.95.2 that we use in our setup is not capable of supporting all processors shown in Appendix A, *Supported Processors and Evaluation Platforms*. The ARM Thumb, Hitachi SH, MN10300/AM33, NEC MIPS VR4300, and NEC V850 processors require installation of a snapshot of the GNU C/C++ Compiler. The snapshots can be found online at <ftp://gcc.gnu.org/pub/gcc/snapshots>

The latest versions of the GNU cross-development tools, as well as additional documentation for each group of tools, can be found on their respective home sites:

- GNU Binary Utilities—<http://sources.redhat.com/binutils>
- GNU C/C++ Compiler—<http://gcc.gnu.org>
- GNU Insight Debugger with Insight Interface—<http://sources.redhat.com/insight>

When installation is complete, the cross-development tools are found in the `D:\cygwin\tools\H-i686-pc-cygwin\bin` subdirectory.

The discussion mailing list and address for posting for the GNU Binary Utilities can be found online:

- Discussion Mailing List—<http://sources.redhat.com/ml/binutils>
- Post to—[binutils@sources.redhat.com](mailto:binutils@sources.redhat.com)

The discussion mailing list and address for posting for the GNU C/C++ Compiler can be found online:

- Discussion Mailing List—<http://gcc.gnu.org/ml/gcc>
- Post to—[gcc@gcc.gnu.org](mailto:gcc@gcc.gnu.org)
- Additional Discussion Mailing Lists—<http://gcc.gnu.org/lists.html>

The discussion mailing lists and addresses for posting for the GNU Insight Debugger (and GDB) can be found online:

- Insight Debugger Discussion Mailing List—<http://sources.redhat.com/ml/insight>
- Post to—[insight@sources.redhat.com](mailto:insight@sources.redhat.com)



- GDB Discussion Mailing List—<http://sources.redhat.com/ml/gdb>
- Post to—[gdb@sources.redhat.com](mailto:gdb@sources.redhat.com)
- Additional GDB Discussion Mailing Lists—<http://sources.redhat.com/gdb/mailling-lists>

### STEP 1

Open the bash command shell. This can be done by clicking on the Cygwin shortcut on the desktop, if created in the Cygwin native tools installation, or through the menu *Start -> Programs -> Cygnus Solutions -> Cygwin Bash Shell*.

When the shell is opened, the present working directory is `D:\cygwin\home\xxx`, where `xxx` is your username. We want to change to the root Cygwin directory by entering the following command at the bash prompt:

```
$ cd /
```

### STEP 2

Unzip the i386 GNU cross-development tools into our Cygwin directory structure. The command for this is:

```
$ tar xjvf /cygdrive/e/gnu/i386gntools.tar.bz2
```

---

**NOTE** The CD-ROM drive is mounted as `/cygdrive/e/` by default when Cygwin is installed. The drive letter for your CD-ROM should be substituted in place of `/e/` in the preceding command. If you are uncertain of the drive mountings for your system, enter the command `mount` at the bash shell prompt to get a listing of the current system mounts.

After executing this command, the i386 GNU cross-development tools are located under the `D:\cygwin\tools` directory. The binary executables are under the `D:\cygwin\tools\H-i686-pc-cygwin\bin` directory.

### STEP 3

We need to ensure that the Cygwin temporary directory is mounted. The command to do this is:

```
$ mount -f -b d:/cygwin/tmp /tmp
```

### STEP 4

Next, we set the path for our new GNU cross-development tools. The bash shell command for this is:

```
$ PATH=/tools/H-i686-pc-cygwin/bin:$PATH ; export PATH
```

We also add the GNU cross-development tools directory to the Windows environment path. The path is altered by right-clicking on the My Computer icon on the desktop. This brings up a drop-down list of options. Select Properties from the drop-down list.

The System Properties dialog box is displayed. Select the Environment tab. Under the User Variables, select path. In the Value edit box, to the front of the path, add:

```
D:\cygwin\tools\H-i686-pc-cygwin\bin;
```

Then, click the Set button. Finally, click the OK button.

### STEP 5

We can verify that the i386 GNU cross-development tools were installed properly by entering the command:

```
$ i386-elf-gcc --version
```

The following message is output if everything is set up properly:

```
2.95.2
```

If the message is incorrect, you need to verify that the tools were unzipped correctly and located in the correct directory.

## 10.2.3 Installing the eCos Development Kit

There are multiple phases for the eCos development kit installation. The files needed for this installation are located under the `ecos` directory on the CD-ROM. We are going to install the eCos development kit into the `D:\ecos` directory.

The file `ecos-v2a-snap.tar.bz2` contains the source code files from a snapshot of the version 2 release of the eCos source repository. Also included in this file are the eCos configuration tools.

During this installation procedure, we install the version 2 Configuration Tool and the version 1.3.net version as well. To build our RedBoot and eCos images, as we will in Chapter 12, we use the version 2 Configuration Tool. The version 1.3.net Configuration Tool is installed for using the Memory Layout Tool (MLT), which is described in Chapter 11.

Also included in this installation is the Package Administration tool and the eCos command-line configuration tool.

The Linux installation files are located on the CD-ROM under the `ecos\linux` directory. Additional information about installing eCos on a Linux host platform can be found online at:

<http://sources.redhat.com/ecos/getstart.html>

### STEP 1

Open a bash shell and change to the root `D:\` drive by entering the command:

```
$ cd d:
```

Now we unzip the source code files by entering the command:

```
$ tar xjvf /cygdrive/e/ecos/ecos-v2a-snap.tar.bz2
```

After the files have been unzipped, we have a new directory, `ecos`, containing the source code repository and other eCos development kit files.

---

**NOTE** The CD-ROM drive is mounted as `/cygdrive/e/` by default when Cygwin is installed. The drive letter for your CD-ROM should be substituted in place of `/e/` in the preceding command. If you are uncertain of the drive mountings for your system, enter the command `mount` at the bash shell prompt to get a listing of the current system mounts.

## STEP 2

Next, we need to install the eCos toolset. The eCos toolset is installed individually. The eCos toolset files are contained under the `D:\ecos\bin` directory, which we just unzipped. First, we install version 2.11 of the eCos Configuration Tool.

It is a good idea to read the two text files that accompany the Configuration Tool version 2.11, which are `readme_cfg_v211.txt` and `changes_cfg_v211.txt`. The file `readme_cfg_v211.txt` contains information about the version 2 release of the Configuration Tool. The file `changes_cfg_v211.txt` includes the modifications for each version, up to the present release, of the Configuration Tool version 2.

To install version 2.11 of the Configuration Tool we run the setup file `configtool-2.11-setup.exe`, located under the `D:\ecos\bin` directory.

The first dialog box asks if we want to install the eCos Configuration Tool 2.11. Click Yes to continue with the install.

## STEP 3

Next, the welcome dialog box for installing the eCos Configuration Tool version 2.11 is displayed. Click the Next button to continue with the installation.

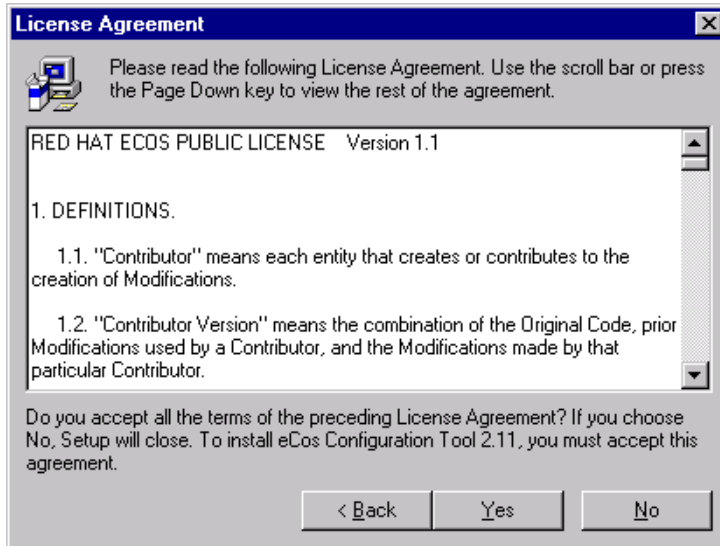
## STEP 4

Now the eCos license agreement dialog box is displayed, as shown in Figure 10.11.

Use the scroll bar on the right side of the dialog box to view the entire license agreement. Click Yes to accept the license agreement and continue with the installation.

## STEP 5

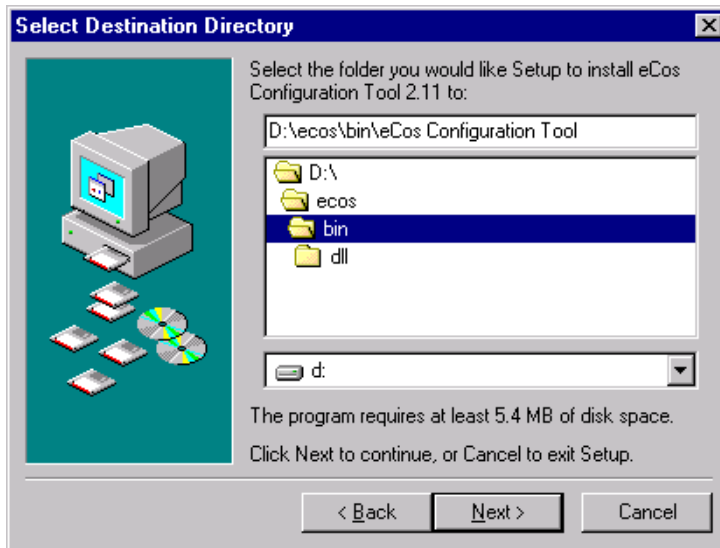
Next, the Configuration Tool version 2 readme file is displayed in a dialog box. The readme file is also included in the `D:\ecos\bin` directory in the file `readme_cfg_v211.txt`. Use the scroll on the right side of the dialog box to view the entire readme file. Click the Next button to continue.



**Figure 10.11** eCos Configuration Tool license agreement dialog box.

## STEP 6

Now we need to select a location for the destination of the Configuration Tool files. The destination selection dialog box is shown in Figure 10.12.

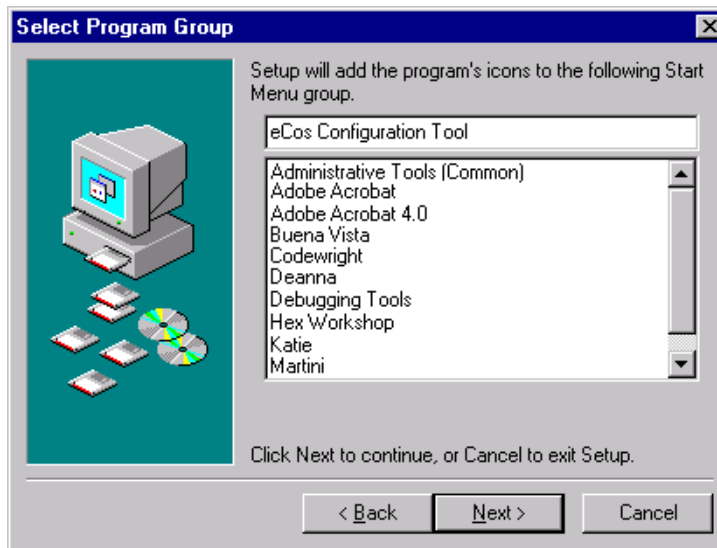


**Figure 10.12** Configuration Tool destination directory selection dialog box.

Select `D:\ecos\bin\eCos Configuration Tool` for the destination directory. Use the Browse button to select the `D:\ecos\bin` directory and the setup file appends the `eCos Configuration Tool` portion of the directory. Then, click the Next button to continue.

### STEP 7

Next, we select the program group to install the Configuration Tool start menu shortcuts, as we see in Figure 10.13.



**Figure 10.13** Configuration Tool program group selection dialog box.

We enter `eCos Configuration Tool` into the edit box, as shown in Figure 10.13. Then, click the Next button to continue.

### STEP 8

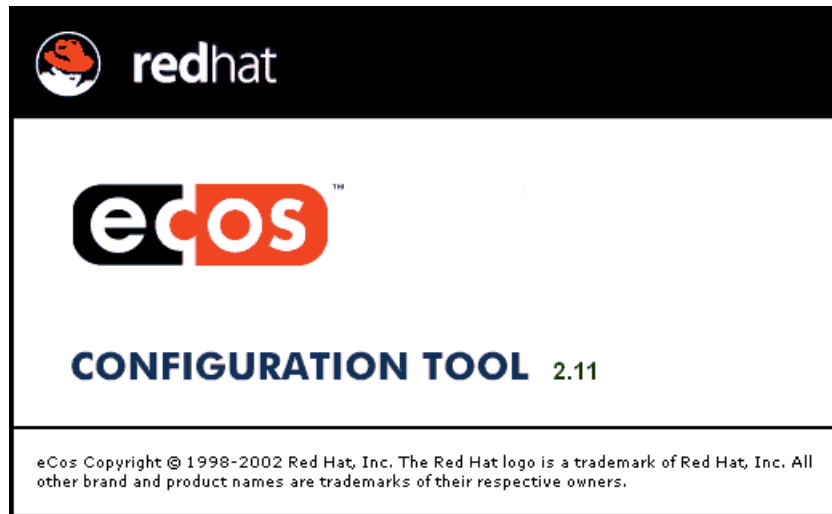
A dialog box showing that setup is ready to begin the installation of files is now displayed. Click the Install button to start the installation.

A progress bar is displayed during the installation procedure. When the installation is complete, a dialog box asking to reboot the computer is displayed. Select Yes, restart the computer now, and click the Finish button to complete the Configuration Tool installation. After the computer reboots, we see the Configuration Tool icon on our desktop.

### STEP 9

Double-click on the Configuration Tool icon on the desktop to run the `eCos Configuration Tool`. We could also select Configuration Tool from the *Start -> Programs -> eCos Configuration Tool* menu.

The Configuration Tool version 2.11 splash screen is displayed, as shown in Figure 10.14. The version of the Configuration Tool is located on the splash screen.



**Figure 10.14** Configuration Tool version 2.11 splash screen.

The program displays a dialog box asking for the location of the eCos repository tree. We click the Browse button and select the `D:\ecos` directory. Then click OK. The eCos repository tree location can be changed later by selecting *Build* → *Repository* from the menu.

The Configuration Tool then searches through the `ecos.db` file under the `D:\ecos\packages` directory to display the appropriate package information. You can find additional details about the `ecos.db` file in Chapter 11.

#### **STEP 10**

Next, we set up the build and user tools that the Configuration Tool uses to build eCos and Red-Boot images. Select *Tools* → *Paths* → *Build Tools* from the menu. We want to browse to the `D:\cygwin\tools\H-i686-pc-cygwin\bin` directory and click the OK button.

Now we set the user tools by selecting *Tools* → *Paths* → *Build Tools* from the menu. We browse to the `D:\cygwin\bin` directory and click OK.

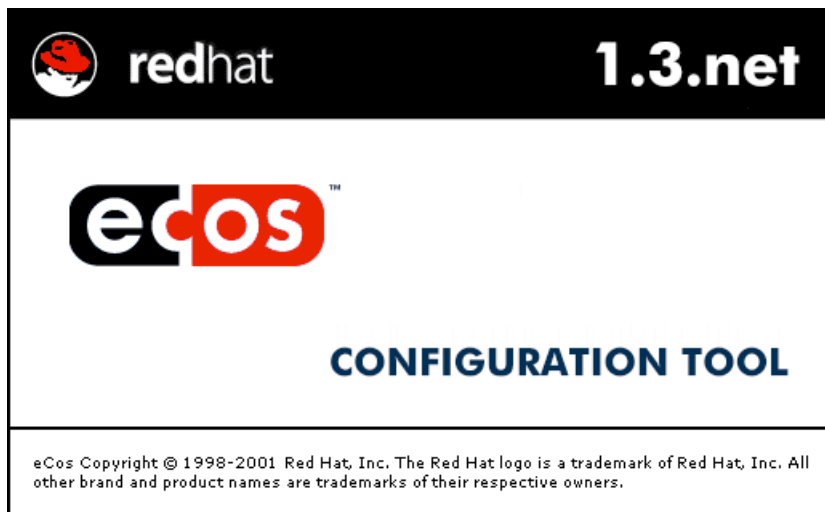
We now close down the Configuration Tool version 2.11 and finish installing the other eCos development tools.

#### **STEP 11**

Continuing with the eCos development kit installation, we install a registry file to enable us to run the eCos Configuration Tool version 1.3.net. We only need to use version 1.3.net of the Configuration Tool when we need to use the MLT. Otherwise, we must hand edit the memory files associated with our configuration.

To update the registry, double-click on the file `eCos-011025.reg`, which is located in the `D:\ecos\bin` directory. A dialog box notifying that the registry has been updated is displayed. Click OK to close the dialog box.

We can now run the Configuration Tool version 1.3.net by running the file `Configtool13net.exe`, which is also located in the `D:\ecos\bin` directory. The version 1.3.net splash screen is displayed, as shown in Figure 10.15. Notice the version of the Configuration Tool located in the upper right-hand corner of the splash screen.



**Figure 10.15** Configuration Tool version 1.3.net splash screen.

The readme file for version 1.3.net of the Configuration Tool is located in the `D:\ecos\bin` directory in the file `readme_cfg_13net.txt`.

## STEP 12

Finally, let's ensure that we can run the Package Administration tool and the eCos command-line configuration tool.

First, run the Package Administration tool by running the file `PkgAdmin.exe` under the `D:\ecos\bin` directory.

---

**NOTE** The Package Administration tool requires certain DLLs to run properly. These DLLs are contained in the directory `D:\ecos\bin\dll`. They need to be copied into the `C:\windows\system` directory to run the Package Administration tool. However, they should only be copied to this directory if they do not exist or if older versions exist in this Windows system directory. It is a good idea to make a backup copy of the DLL files being replaced prior to replacing these files.

Click the Close button to exit the Package Administration tool. Additional information about the eCos Package Administration tool is provided in Chapter 11.

Next, open a bash shell, if one is not open. To see if the eCos command-line tool runs properly, enter the command:

```
$ d:/ecos/bin/ecosconfig.exe
```

The command line configuration tool usage output should be displayed in the bash shell. This information can also be displayed by entering `ecosconfig --help`. Additional details about the command-line configuration tool are included in Chapter 11. If you plan to use the command-line configuration tool often, it is a good idea to add `D:\ecos\bin` to the path.

### 10.2.3.1 eCos Development Kit Directory Structure

Let's take a look at the eCos development kit directory structure. The first directory is `bin`, which contains the eCos toolset executable files.

The `CVS` directory is set up when we access the eCos online source code repository. We go through this process in the following section.

Next is the `doc` directory. This contains Standard Generalized Markup Language (SGML) files of the most recent eCos documentation. These files are contained in the eCos online repository.

The `examples` directory includes several basic eCos sample applications. We get into example applications later in Chapter 12.

The `host` directory includes the source code for all of the tools in the eCos toolset, including the Configuration Tool and Package Administrator.

The `packages` directory contains the version 2 snapshot of the eCos online source code repository.

### 10.2.4 Accessing the Online eCos Source Code Repository

Due to the constant changes, contributions, and bug fixes made to the eCos source code, it will quickly become necessary to take advantage of these modifications and update your local source code repository. It is very important to keep up to date with the latest bug fixes and enhancements to the eCos, and RedBoot, source code. In order to do this, we must configure our system to access the remote eCos repository.

The eCos source code repository, which includes RedBoot, is managed by CVS, and can be accessed publicly without requiring an account on the remote machine, which is termed anonymous CVS access. Additional information about CVS can be found online at:

[www.cvshome.org](http://www.cvshome.org)

Additional information about eCos access to the source code repository using anonymous CVS can be found online at:

<http://sources.redhat.com/ecos/anoncv.html>



A popular Linux CVS client called TkCVS is also included on the CD-ROM under the `tkcvs` directory. Information about TkCVS as well as installation instructions can be found online at:

[www.twobarleycorns.net/tkcv.html](http://www.twobarleycorns.net/tkcv.html)

#### 10.2.4.1 Installing WinCVS

To access the eCos source code repository we use one of the open-source CVS software packages called WinCVS (version 1.2). The files needed for installing WinCVS are located on the CD-ROM under the `wincvs` directory. There is one basic setup executable to run that guides us through the installation procedure.

The source code files, `WinCvs120_src_app.zip` and `WinCvs120_src_shared.zip`, for the WinCVS version used in this installation are also contained in the `wincvs` directory in Windows zip file format. The WinCVS home site is located at:

[www.cvsgui.org](http://www.cvsgui.org)

#### STEP 1

Run the WinCVS setup file `Setup.exe`. The first dialog box displayed is the welcome dialog box. Click the Next button to proceed with the installation.

#### STEP 2

Next, the software license agreement dialog box is displayed, as shown in Figure 10.16.

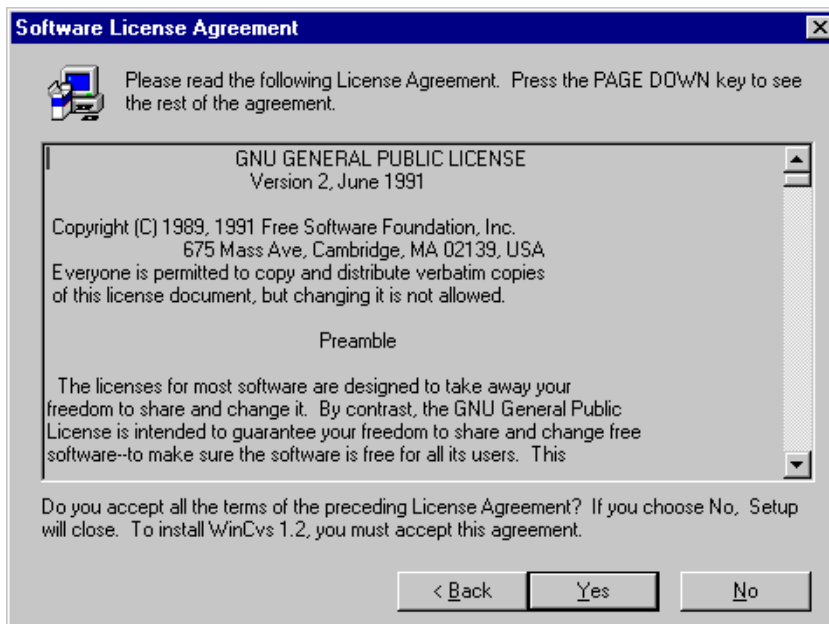
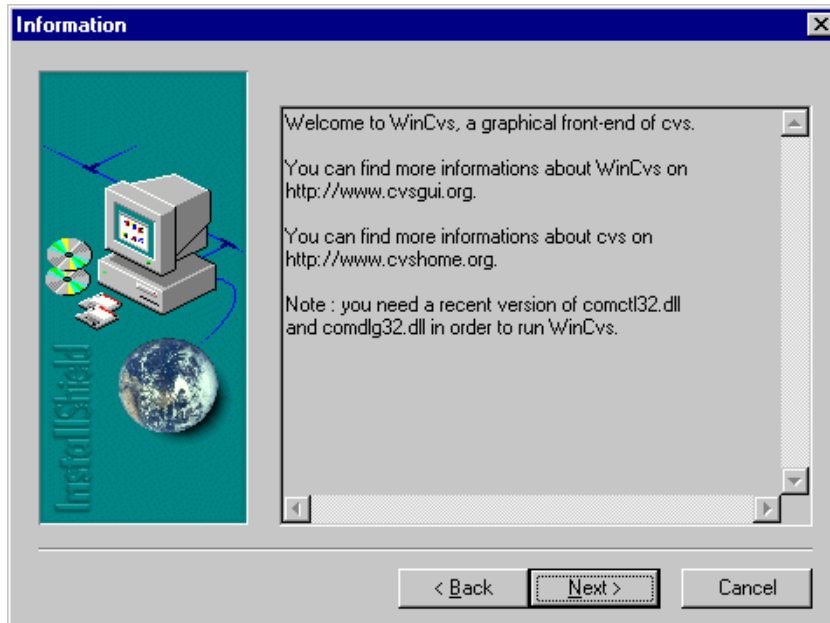


Figure 10.16 WinCVS Software License Agreement dialog box.

WinCVS is also licensed under the GPL version 2, as we see in Figure 10.16. Click the Yes button to accept the license agreement and continue.

### STEP 3

The next dialog box is the information dialog box, as shown in Figure 10.17.

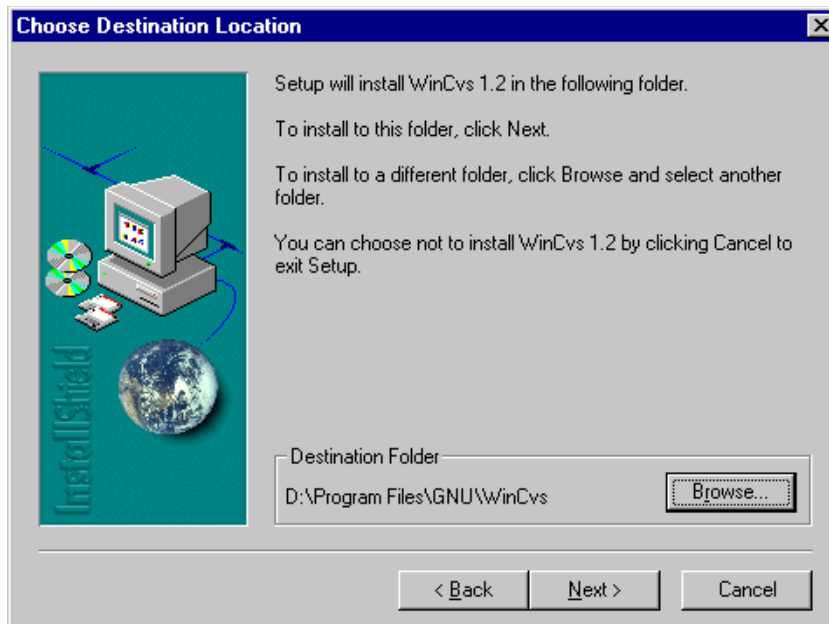


**Figure 10.17** WinCVS information dialog box.

The information dialog box displays the WinCVS home site, the CVS home site, and additional information about the DLL files needed to run WinCVS. Click the Next button to continue.

**STEP 4**

Next, we select the destination location to install WinCVS. The dialog box for this step is shown in Figure 10.18.

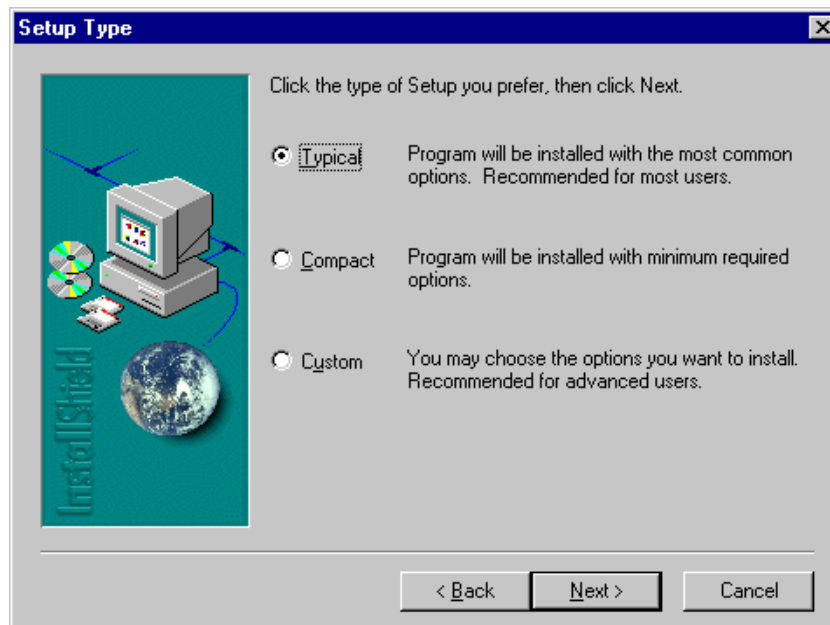


**Figure 10.18** WinCVS destination location dialog box.

The destination selected is `D:\Program Files\GNU\WinCVS`. Click the **Browse** button to enter this directory. Click the **Next** button to continue.

**STEP 5**

Now we select the type of installation. The possible choices are shown in Figure 10.19.

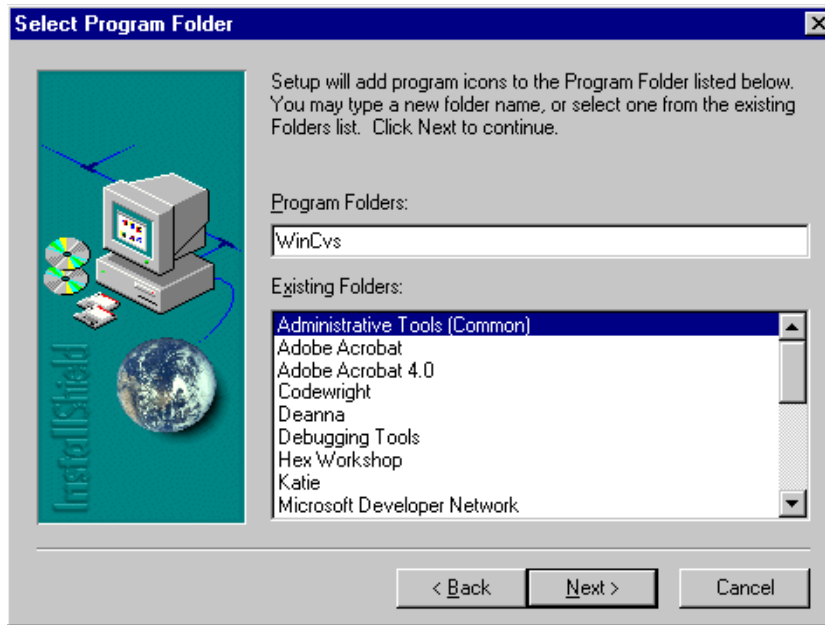


**Figure 10.19** WinCVS Installation Type dialog box.

The installation type we want is Typical, which is the default. Click the Next button to continue.

**STEP 6**

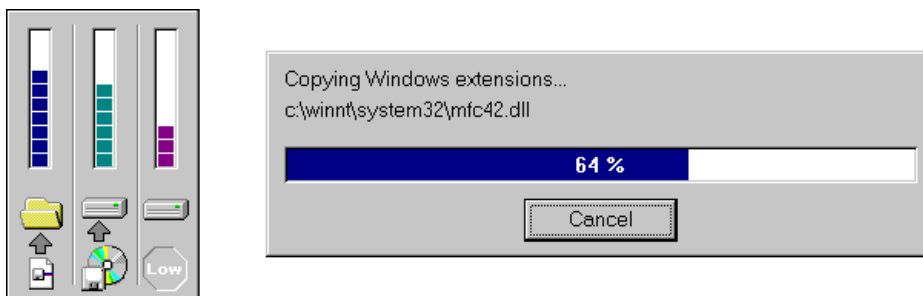
Next, we select the program folder for the WinCVS icons. This dialog box is shown in Figure 10.20.



**Figure 10.20** WinCVS Program Folder dialog box.

The program folder entered is WinCVS. Click the Next button to continue. Then, the setup program is ready to begin the installation by displaying the Start Copying Files dialog box. Click Next to start the installation.

As the setup continues, icons similar to the ones shown in Figure 10.21 display the progress of the installation.

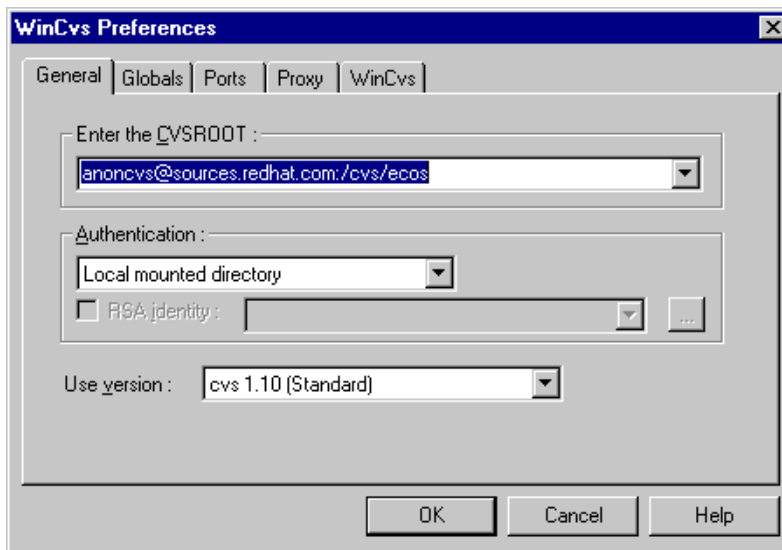


**Figure 10.21** WinCVS progress icons.

After the installation is complete, the computer needs to be rebooted. Select *Yes, I want to restart my computer now*, which is the default, and click the Finish button in the Setup Complete dialog box.

#### 10.2.4.2 Setting WinCVS Preferences

After the computer reboots, we can launch WinCVS using the shortcut for the WinCVS executable located under *Start -> Programs -> WinCVS*. Before we can use WinCVS, we must configure it for the eCos source code repository. The dialog box shown in Figure 10.22 is displayed the first time we launch WinCVS.



**Figure 10.22** WinCVS preferences dialog box.

In the preferences dialog box, under the Enter the CVSROOT edit box we want to type in:

```
anoncvs@sources.redhat.com:/cvs/ecos
```

Then we select *Local mounted directory* from the Authentication drop-down list.

Next, *cvs 1.10 (Standard)* is selected from the Use version drop-down list.

Then we click on the Globals tab. Since we want the capability to modify the eCos source files after we download them from the online repository, we want to uncheck the *Checkout read-only* checkbox.

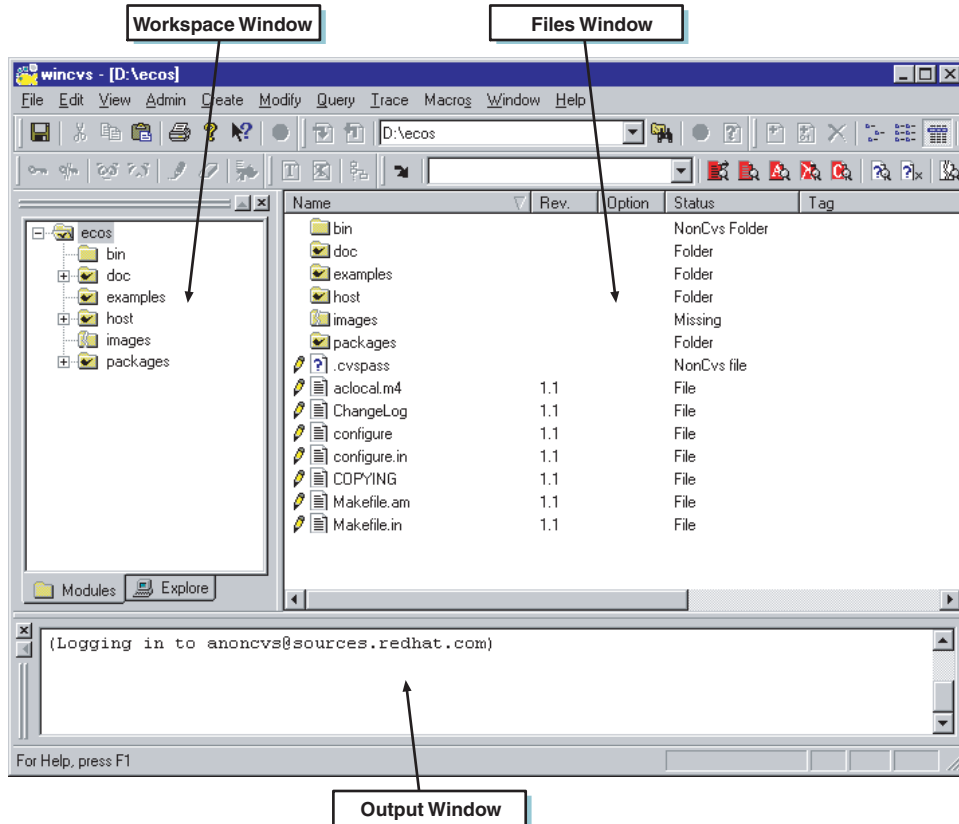
Now we click on the WinCvs tab and enter the HOME folder. To enter this, we browse to the `D:\ecos` directory.

Then, click the OK button to set the preferences.

The preferences can be modified at anytime by selecting *Admin -> Preferences*.

### 10.2.4.3 WinCVS Update Commands

In this section, we take a brief look at some of the common commands used to update the eCos source code repository. After setting the preferences, the main WinCVS program screen is shown, similar to the one shown in Figure 10.23.



**Figure 10.23** WinCVS main program screen.

As we see in Figure 10.23, there are three main windows in WinCVS. On the left is the workspace window. This displays the current home directory, and the directories below, which contain the CVS source code. To the right of the workspace window is the file window, which shows the files and directories located within the home directory. At the bottom is the output window. The output window shows output generated when running WinCVS commands. The output window can also be used to enter commands, as we do next.

In the output window, we enter the command:

```
cvs -d :pserver:anoncvs@sources.redhat.com:/cvs/ecos login
```

This command allows us to log in to the online eCos repository. When the password dialog box comes up, enter:

```
anoncvs
```

The password is stored in the file `.cvspass` under the WinCVS home directory, which we selected during the preferences setup as `D:\ecos`. We are now ready to enter commands to update our local eCos source code repository.

---

**NOTE** Updating your local eCos source code repository should not be done at this point. The examples we go through in Chapter 12 have been developed using the source code repository from the eCos development kit installation procedure. It is a good idea to run through the examples prior to updating to the latest eCos repository source code.

It is a good idea to use different directory trees when checking out snapshots of the eCos source code rather than using a directory containing a full release. To check out the latest snapshot of the eCos repository, create a new directory for checkout. Then, change to that directory. Next, enter the following command in the WinCVS output window:

```
cvs -d :pserver:anoncvs@sources.redhat.com:/cvs/ecos -z 6 co -P ecos
```

This is a checkout (`co`) command. The `-d` option specifies the CVS root directory. The `-z 6` option sets the compression level. `co` is a synonym for the checkout command. The option `-P` notifies WinCVS to prune any empty directories. Finally, `ecos` is the module to checkout.

---

**NOTE** The entire eCos repository, including the source code for the eCos host tools and RedBoot binary images, can be downloaded using `ecos-full` as the module name; for example, `cvs -d :pserver:anoncvs@sources.redhat.com:/cvs/ecos -z 6 co -P ecos-full`. However, be cautious using this because all RedBoot images will be downloaded, which totals about 12 Mbytes.

This initiates a connection to the CVS server and checks out the latest version of eCos, which includes all packages and RedBoot as well.

After a local repository has been created, different commands are entered to update the repository. For example,

```
cvs -z 6 update -d -P
```

The preceding command updates any files that have been modified since the last time the files were checked out.

Other commands are available to check out repository files from specific dates or with specific labels. Additional information about commands is located in the WinCVS help. Select



*Help* → *Help on cvs-1.10*. Select the Index button and type in `command reference`. This gives a listing of the CVS commands and optional switches. There is also information on the eCos site at:

<http://sources.redhat.com/ecos/anoncvs.html>

### 10.3 Summary

We have just completed our installation of the eCos host development tools for Windows, which includes the Cygwin tools, the cross-development tools for the Intel x86 processor, and the eCos development kit.

We also went through the update procedure using the eCos online repository. This enables us to keep up to date with the latest changes, additions, and bug fixes with the eCos and RedBoot source code.

Entering incorrect commands causes some of the most common problems during installation. You should make sure that the commands are entered correctly prior to proceeding to the next step.

Another common problem is insufficient disk space. The amount of space needed varies depending on the format of the hard drive. Verify that an ample amount of hard drive space is available before configuring your host development platform.

If problems arise when installing the tools, a great source for information is the mailing list for the associated tool. Often, the developers on the mailing list have been through the same steps before and can offer helpful hints to getting through snags that might come up.

We are now ready to proceed with the eCos development kit and learn about the different tools available and how we can use them to develop our own applications.

## The eCos Toolset

**T**his chapter begins by looking at the structure of packages in the eCos repository, including a brief description of the CDL used for package script files. We then look at the graphical Configuration Tool used to customize and build the eCos library, and then build and run tests on your target platform.

Finally, we look into the other eCos tools and additional open-source tools to aid in our embedded software development. These additional open source tools allow us to set up a complete software embedded development environment at no cost. This chapter prepares us for building the eCos library with the tools provided in the eCos development kit. Using this information, we can then build an example application, which is covered in the next chapter, and run it on our target hardware platform.

### 11.1 Packages

To get a better understanding of the eCos toolset, we need to take a closer look at the modules that the tools operate on: packages. As described in Chapter 1, *An Introduction to the eCos World*, a *package* is a software component incorporating all necessary source and configuration files for distribution. A package can be for a single hardware device or for an entire networking stack. Packages can be configured, using the different eCos tools, to operate in a mode specific to your application. We see how to manipulate packages with the eCos configuration tools in the *Package Control* section of this chapter.

In order for the eCos component framework to make use of packages, the package must follow rules imposed by the framework. A CDL script file is included with each package, which describes the package to the component framework. Included in the CDL script files are

dependencies for using the package, configuration options and their associated value ranges, and details on how to build the package.

### 11.1.1 Package Directory Structure

All packages in the eCos repository are located under the `packages` subdirectory under the root eCos development kit install directory; in our case, `D:\ecos`. A `packages` directory snapshot is shown in Figure 1.3 in Chapter 1. Because the eCos source code repository is constantly evolving to accommodate new features and hardware platforms, the latest eCos repository might differ from the one shown in Figure 1.3.

As we see in Figure 1.3, the different packages are arranged based on the functionality provided by the package. For example, all device drivers are contained in the `devs` subdirectory, while all compatibility layer support, such as POSIX and  $\mu$ ITRON, is contained under the `compat` subdirectory. The depth of the subdirectory structure is different for each package. It is a good idea to familiarize yourself with the overall directory structure by browsing through your eCos development kit installation.

Under the `packages` directory is the package database file `ecos.db`. This file uses the CDL and contains a high-level description for all packages in the component framework. Additional details about the eCos database file are included later in *The Configuration Tool* section of this chapter.

Although the depth of the different subdirectory structures varies for each package, each of these packages is comprised of a number of common subdirectories that separate the functionality supplied in the different files. The common directory structure for an example package is shown in Figure 11.1. Not all packages contain all of the common package subdirectories shown in Figure 11.1; some packages contain more, some less.

---

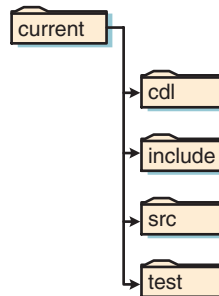
**NOTE** Our local eCos source code repository, installed in Chapter 10, *The Host Development Platform*, also contains `CVS` directories at different levels of our directory tree. These directories are used by the `CVS` program—in our case, `WinCVS`—to keep track of information about our local eCos repository and the online eCos repository. We can ignore these directories at this point since they do not contain any source code files for eCos.

It is also important to note that not all of the packages strictly follow the rules for the location of files, making the possibility of the file organization slightly different from package to package. In this section, we look at the general structure of typical packages.

As we see in Figure 11.1, the root of the typical package directory is the version. The actual name of this subdirectory depends on the repository version installed. From our installation completed in Chapter 10, we have a version directory called `current`.

Having multiple versions for each package allows you to experiment with new versions while maintaining a working version for a particular package. If the new, experimental version of the package does not work in your application, you can always revert to a previous version.

**Figure 11.1** Typical package directory structure example.



Selection of versions for packages is accomplished using the configuration tools, which we get into in the *Package Control* section of this chapter.

Each package contains a file under the version subdirectory called *ChangeLog*. The eCos source code repository maintainers use these files to track the changes for each of the different files within a package. These files are useful to get an overview of changes made to the package to determine if you want to update to a current version to make use of particular bug fixes or enhancements. An example of two change log entries is shown in Code Listing 11.1.

```
1 2001-09-27 Jonathan Larmour <jifl@ecoscentric.com>
2
3 * src/plf_misc.c (hal_platform_init):
4 If not RAM startup, install exception VSRs.
5
6 2000-04-21 Bart Veer <bartv@ecoscentric.com>
7
8 * pkgconf/rules.mak:
9 Fix header file dependencies for testcases.
```

**Code Listing 11.1** Example *ChangeLog* entries.

In Code Listing 11.1, lines 1 and 6 contain the date the changes were made, the developers who made the changes, and the developers' email addresses. Each change is designated with an asterisk (\*) as shown on lines 3 and 8. After the asterisks are the files that were altered and their directory location within the package. Sometimes the routine modified is also entered as we see on line 3, in which case the routine `hal_platform_init` was modified. Lines 4 and 9 describe the changes made.

Under the version root is the `cdl` subdirectory. This subdirectory must be included within a package. This contains at least one CDL script file that defines the package. Some packages, such as the `kernel`, might break up the functionality of the CDL script in multiple CDL files. Additional details about CDL script files are contained in *The Component Definition Language Overview* section of this chapter.

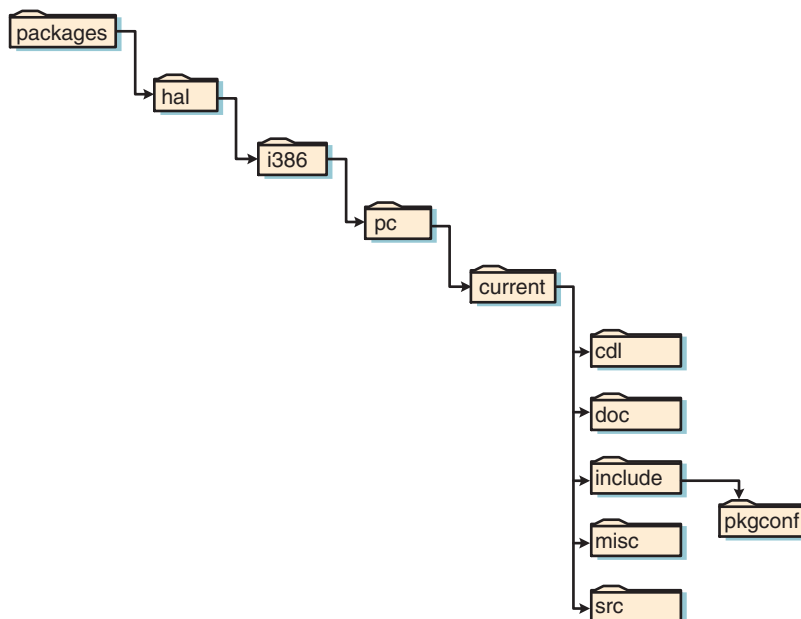
Next is the `include` subdirectory, which contains public header files for the package. During the eCos library build process, which we go through in Chapter 12, *An Example Application Using eCos*, some of the header files for packages that are used are copied into the eCos library working directories. This allows an application to include header files needed for a particular package from the working directory rather than using the repository directories.

The `src` subdirectory contains the source files for the package. This subdirectory must be included within a package. The source files might include `.c` C language files, `.cxx` C++ language files, and `.S` assembly language files. Private header files (with a `.h` extension), accessible only to the package source code, might also be contained in this directory. The `src` subdirectory can be broken down further into additional subdirectories to aid with the organization of the source code. For example, see the `kernel\current\src` and `net\tcpip\current\src` subdirectories.

Finally is the `tests` subdirectory. This subdirectory contains any test files relevant for the package. We look at building and running tests in Chapter 12.

Other subdirectories that are often included under certain packages are `doc`, which contains documentation for the package, and `misc`, which contains other files specific to the package such as RedBoot configuration import files for various startup types.

Let's look at the actual directory structure for one of the packages in the eCos source code repository. Figure 11.2 is a snapshot of the i386 PC HAL package directory structure. From this figure, we can see where the common subdirectories are located for this particular package.



**Figure 11.2** Current version directory snapshot of the i386 PC HAL package.

As mentioned before, the `packages` subdirectory is located under the root eCos development kit install directory `D:\ecos`. Since this is a HAL package, it is located under the `hal` subdirectory. The HAL packages are organized by architecture; in this case, it is the `i386` architecture.

The architecture, in this package, is further divided by platform support. The platform component is located under the `pc` subdirectory. The version subdirectory is called `current`. Finally, we have the common package directory structure comprised of the subdirectories `cdl`, `doc`, `include` (along with `pkgconf`), `misc`, and `src`.

We can see that the `i386` package differs slightly from the common package directory structure shown in Figure 11.1. The additional directories for this package are:

- **doc**—contains documentation and release notes for the package.
- **pkgconf**—includes RAM, ROM, and floppy disk memory configuration files.
- **misc**—contains exported floppy disk and ROM RedBoot configuration files for use with the configuration tools.

### 11.1.2 The Component Definition Language Overview

We now take a brief look at the CDL. This section gives us a basic understanding of the CDL and how script files are used with packages in the component repository. Using the information in this section, we can then proceed to explore the eCos configuration tools and how the package CDL script files are interpreted in *The Configuration Tool* section of this chapter. An in-depth look into the CDL is provided online in the *eCos Component Writer's Guide*:

<http://sources.redhat.com/ecos/docs.html>

Details about the CDL command and keyword syntax are in the *CDL Language Specification* section of the *eCos Component Writer's Guide*.

CDL is not a new language, it is an extension of the existing Tool Command Language (Tcl) scripting language. The CDL syntax uses Tcl syntax. The CDL is a key part of the eCos component framework that is used in script files (with the extension `.cdl`) to describe the package to the framework. These CDL script files contain information on all of the configuration options within a package, as well as details on how to build the package.

#### 11.1.2.1 CDL Script Files

Each package has a single top-level CDL script file, typically located in the `cdl` subdirectory, as shown in Figure 11.1. The first command in the top-level script should be `cdl_package`, which is used in the package database file `ecos.db`. There should only be one `cdl_package` command per package.

The CDL syntax implements a command hierarchy. This allows options to be controlled as a group; therefore, disabling a single component disables all options within the component as well. The hierarchy also enables a simpler representation in the Configuration Tool for navigation and modification of options, as we see in *The Configuration Tool* section of this chapter.

There are four CDL commands used in script files:

- **cdl\_package**—The unit of distribution, which might contain option, component, or interface commands within its body. Typically, this is the top level of the CDL script file. The package command body can make use of most option properties as well as additional properties or commands that apply to the entire package.
- **cdl\_component**—A configuration option that might contain additional options or subcomponents within its body. The body of a component command might also contain the same properties as the option command.
- **cdl\_option**—The basic unit of configurability that generally contains a single choice. The body of the option command contains properties that describe and define the characteristics of the option. There might also be information to aid in your decision, such as a text description of the option.
- **cdl\_interface**—A calculated configuration option that provides an abstraction mechanism. The body of the interface command might contain a subset of the option properties.

Every command must have a command name. All command names must be unique within a configuration. If two names exist within two different packages, it is not possible to load both packages in a configuration.

---

**NOTE** You might notice that some packages use the same CDL command names. For example, all HAL packages define the CDL component name `CYG_HAL_STARTUP`. Even though all command names must be unique within a configuration, this is fine because only one HAL package can be loaded at a time; therefore, the name `CYG_HAL_STARTUP` only exists once within a configuration.

Typically, the component framework outputs a `#define` for every active and enabled option, using the command name as the symbol being defined. For example, enabling the option command named:

```
CYGIMP_KERNEL_INTERRUPTS_DSRS
```

which is located within the eCos Kernel package, in the Kernel Interrupt Handling component causes

```
#define CYGIMP_KERNEL_INTERRUPTS_DSRS 1
```

to be placed in the designated kernel package header file. Some options are set to values in which case the configured value is used when setting the `#define` statement.

Command names in the eCos component repository follow a naming convention to avoid name clashes with other packages. An example of a package name is `CYGPKG_HAL_I386_PC`.

The first three characters of the name identify the organization that produced the package; in this case, CYG is for Cygnus Solutions. The next three characters indicate the nature of the option; in this case, PKG is used to indicate this is a package. A complete list of command tags is given in *The eCos Component Writer's Guide*. The HAL portion of the name indicates the location of the option within the overall hierarchy; in this case, we see that this is a HAL package. The last part, I386\_PC, indicates the option itself. This is for the i386 architecture PC platform package.

Each CDL command contains a body that might include properties or commands that describe to the component framework how to handle each option. Properties can be descriptions of the command or values set by you that are used when the package is built. Commands within the CDL command body can define header files generated for the package, list source files to build for the package, or identify constraints that must be met if the package is active. The basic CDL command structure is shown in Code Listing 11.2.

```
1 cdl_package PACKAGE_NAME {
2 <package properties and commands>
3 cdl_component COMPONENT_NAME {
4 <component properties and commands>
5 }
6 cdl_option OPTION_NAME {
7 <option properties and commands>
8 }
9 }
```

**Code Listing 11.2** Basic CDL command structure example.

As we see in Code Listing 11.2, a basic structure for CDL commands contains the command followed by the command name, as shown on line 1. In this case, the package command name is `PACKAGE_NAME`. Every command body is encapsulated in curly brackets (`{ }`). The `#` character is used for comments within the CDL script file. Line 2 is the beginning of the package command body and the package command ends on line 9. Package commands and properties are located within the package body. There can be any number of package commands, properties, options, or components in the package body.

Next is a component command on line 3, named `COMPONENT_NAME`. Within the component command body, shown on line 4, there might be a number of different properties, commands, options, or subcomponents present.

Finally, there is an option command named `OPTION_NAME` on line 6. The body of the option command is on line 7 and might contain any number of properties and commands to describe and define the option.

Code Listing 11.3 shows an excerpt from the i386 PC HAL package. The script file for this listing is `hal_i386_pc.cdl` located under the `hal\i386\pc\current\cdl` directory. However, some of the script commands and properties have been cut out to reduce the size of the listing and isolate the commands we want to examine.



---

**NOTE** The CDL script fragment in Code Listing 11.3 does not show all of the possible commands or properties available. This fragment is intended to familiarize us with some of the basic language of the CDL and the structure of script files. In the *Graphical Representation of CDL Script Files* section of this chapter we see how the component framework and configuration tools use packages and their CDL script files. For a comprehensive look at the CDL, read *The eCos Component Writer's Guide*. This document is very useful if you need to develop your own packages for distribution.

```
1 cdl_package CYGPKG_HAL_I386_PC {
2 display "i386 PC Target"
3 parent CYGPKG_HAL_I386
4 define_header hal_i386_pc.h
5 include_dir cyg/hal
6 description "
7 The i386 PC Target HAL package provides
8 the support needed to run eCos binaries
9 on an i386 PC."
10
11 compile hal_diag.c plf_misc.c plf_stub.c
12
13 implements CYGINT_HAL_DEBUG_GDB_STUBS
14 implements CYGINT_HAL_DEBUG_GDB_STUBS_BREAK
15 implements CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT
16
17 cdl_component CYG_HAL_STARTUP {
18 display "Startup type"
19 flavor data
20 legal_values {"RAM" "FLOPPY" "ROM"}
21 default_value {"RAM"}
22 no_define
23 define -file system.h CYG_HAL_STARTUP
24 description "
25 It is possible to configure eCos
26 for the PC target to build for RAM
27 startup (generally when being run
28 under an existing monitor program
29 like RedBoot), FLOPPY startup (for
30 writing to a floppy disk, which can
31 then be used for booting on PCs with
32 a standard BIOS), or ROM startup
33 (for writing straight to a boot
34 ROM/Flash). ROM startup is experimental
35 at this time."
36 }
```

```
37
38 cdl_option CYGSEM_HAL_I386_PC_DIAG_SCREEN {
39 display "Output to PC screen"
40 flavor bool
41 default_value 1
42 implements CYGINT_HAL_I386_PCMB_SCREEN_SUPPORT
43 description "
44 This option enables use of the PC screen
45 and keyboard as a third virtual serial
46 device."
47 }
48 }
```

**Code Listing 11.3** Example CDL script file from i386 PC HAL package.

Let's take a closer look at the CDL script file excerpt shown in Code Listing 11.3. On line 1, which is the first CDL command in the `hal_i386_pc.cdl` script file, is the CDL package command with the name `CYGPKG_HAL_I386_PC`. From this name, we can see that this is the i386 architecture PC platform HAL package. The display name of the package is given on line 2 with the `display` command.

Next, line 3 contains the `parent` command, which allows this package to be nested under the i386 architecture defined by the name `CYGPKG_HAL_I386`. The `define_header` command on line 4 identifies the header file, `hal_i386_pc.h`, which is generated for this package, and `include_dir` on line 5 is the location where this header file is placed in the local configuration workspace. We look at the local configuration workspace in Chapter 12 when we build the eCos library.

Line 6 contains the `description` command, which is used by the Configuration Tool to give a text description of the package contents. The `compile` command on line 11 lists the files that should be built for this package. The `implements` commands on lines 13, 14, and 15 describe the general interfaces that are included by selecting the i386 PC package.

Next, is a CDL component command on line 17 named `CYG_HAL_STARTUP`. On line 19 is the `flavor` command, which specifies the nature of the component; in this case, `data`. By designating the `flavor` as `data`, when the `#define` is generated for this component the value is set to one of the `legal_values`, as shown on line 20. The values that this component can be set to are `RAM`, `FLOPPY`, or `ROM`. If no change is made to the `CYG_HAL_STARTUP` component, the `default_value` `RAM` is used from line 21. The command `no_define` on line 22 suppresses the normal generation of preprocessor `#define` symbols in the configuration header file. Additional `#define` symbols that go into the configuration header file are specified by the `define` command on line 23.

The final CDL option command on line 38 is named `CYGSEM_HAL_I386_PC_DIAG_SCREEN`. This option is of type `bool` as we can see from the `flavor` command on line 40, which means that the option is either enabled or disabled. The default for this command is enabled, as shown on line 41 with the `default_value` of 1.

## 11.2 The Configuration Tool

There are two tools you can use to configure the eCos repository according to your specific requirements. As mentioned in Chapter 1, the configuration tools give you the ability to customize the eCos library to meet your specific application needs through source-level configuration. You can use the command-line tool or the graphical Configuration Tool.

The command-line tool does not offer the ability to resolve conflicts interactively. The Configuration Tool provides interactive conflict resolution, as well as an easy-to-use interface for configuration setup and management, the facility to build the eCos RTOS library, and the capability to build and run tests on a target platform. We are going to focus on the use of the Configuration Tool for manipulating our configurations throughout this book. If you would like to become more familiar with the command-line tool, additional information can be found online in the *eCos User's Guide*:

<http://sources.redhat.com/ecos/docs.html>

The eCos development tools are located in the `D:\ecos\bin` directory from our installation complete in Chapter 10. The source code for all of the eCos development tools is located under the `D:\ecos\host` directory.

There are two different versions of the Configuration Tool we installed, 1.3.net and 2.11. The version 1.3.net of the Configuration Tool only supports host development environments running the Windows operating system. Version 2.11 of the Configuration Tool can run on Windows and Linux host development systems.

The major difference between the two versions is that the 1.3.net version includes a graphical window called the Memory Layout Tool (MLT) for manipulating memory configurations. Since the MLT functionality is expected to move into a separate application (at sometime in the future), it is not included in the version 2.11 release of the Configuration Tool.

In this book, we use version 2.11 of the Configuration Tool for configuring the eCos framework and building eCos images. We also cover using the MLT, in version 1.3.net of the Configuration Tool, later in this chapter. Both versions of the Configuration Tool were installed in Chapter 10.

We also get a basic understanding of the relationship between the Configuration Tool and the component repository in this chapter. CDL script files are included with all packages, which allow the Configuration Tool to interpret and display the proper information about a given package.

The Configuration Tool can be invoked by either using the desktop icon, if created during the installation procedure, or through the *Start -> Programs -> eCos Configuration Tool* menu.

### 11.2.1 Screen Layout

The first step to understanding the Configuration Tool is to become familiar with the screen layout. Figure 11.3 is a screen snapshot of the Configuration Tool with all window views enabled. All of the windows in the Configuration Tool can be sized according to your needs by adjusting the splitter bars for a given window.

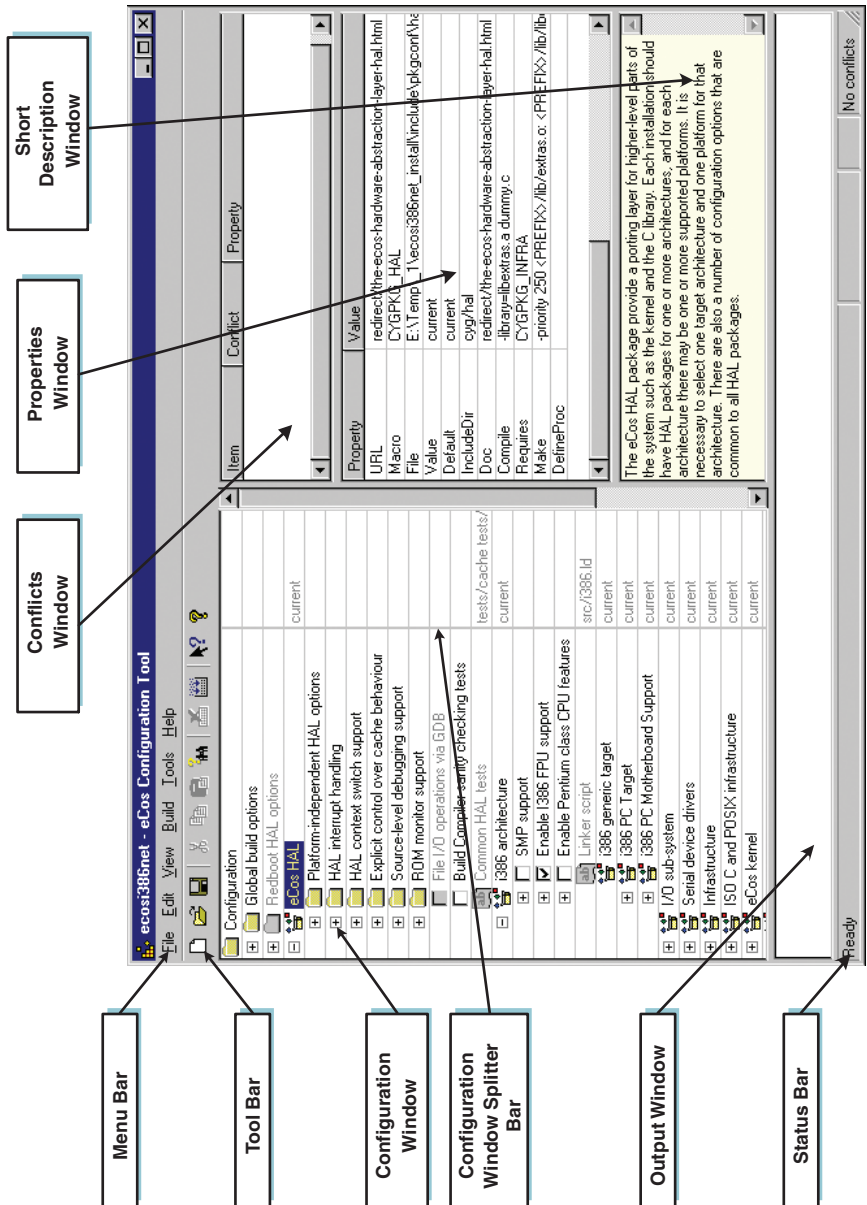


Figure 11.3 Configuration Tool screen layout.

The title bar of the Configuration Tool application displays the configuration file name—in this case, `ecos1386net.ecc`—along with the application name, eCos Configuration Tool. When a modification has been made to the configuration, an asterisk (\*) appears after the configuration filename. This indicates that the modifications made to the configuration have not been saved.

Under the title bar we see the menu bar, which is standard on Windows applications. This contains the menus *File*, *Edit*, *View*, *Build*, *Tools*, and *Help*. The *File* menu contains the options to save, open, start a new configuration, and import or export a configuration.

The *Edit* menu includes the cut, copy, and paste options as well as a find option that allows you to search through certain windows for specific words.

The *View* menu includes options for configuring the settings of the Configuration Tool, control over which windows are displayed, and control over which tool bars are enabled.

The *Build* menu contains options to build the eCos library and tests, clean the build tree working directory, stop a build in progress, display compiler and linker flags, select the root repository, and enable templates and packages for a particular configuration.

The *Tools* menu allows you to set up the paths for the build and user tools, display the available platforms, and resolve conflicts.

Finally, the *Help* menu includes the help documentation for the Configuration Tool and eCos, as well as information about the release of the Configuration Tool. Context-sensitive help topics are displayed for any control within a dialog box by either pressing *F1* or clicking the question mark icon in the dialog caption bar and then clicking the control. Some dialog boxes contain a help button as well.

You might want to customize the Configuration Tool settings according to your needs. To do this, select *View* → *Settings* from the menu bar. This brings up the Configuration Tool settings dialog box. The configuration settings in this dialog box are separated into three different tabs.

The first tab is *Display*. Under this tab, the *Configuration Pane* settings allow you to *Use Macro Names* or *Use Descriptive Names* for the *Labels*. Typically, it is easier to use the descriptive names for a better explanation of the different labels in the Configuration window and refer to the macro name in the Properties window. You can also set whether you want *Integer Items* displayed in *Decimal* or *Hexadecimal*. The *Font* can also be set for the different windows in the Configuration Tool. Finally, the splash screen can be enabled or disabled using the *Show Initial Splash Screen* check box.

Next is the *Viewers* tab. This allows you to set the program to use for viewing header files, which can be set to use the *Associated Viewer* (`wordpad.exe`) or a program you select as *This Viewer*. The program to use for viewing documentation can also be configured to the *Built-in Viewer*, *Associated Browser* (`iexplore.exe`), or a program you select as *This Viewer*.

The *Conflict Resolution* tab is next. This allows you to set the methods the Configuration Tool uses to check for conflicts in your configuration. The available options are *After Any Item Changed*, *Before Saving Configuration* and *Automatically Suggest Fixes*.

Finally is the *Run Tests* tab. The Configuration Tool automates the downloading and running of tests on the target hardware. This tab allows you to configure the timeout associated with

the test download and the connection method to the target, which can either be via serial or Ethernet port. Additional information about running tests using the Configuration Tool is covered in Chapter 12.

The Configuration Tool offers a search tool located under the *Edit -> Find* menu, or the Find icon on the standard tool bar can be used. The Find tool allows you to search for a text string within a configuration item. The Find dialog box allows you to search in macro names, item names, short descriptions, current values, or default values. This tool is very useful when you need to locate a specific configuration item nested deep within the component repository.

Beneath the menu bar is the tool bar. The tool bar shown in Figure 11.3 is the *Standard* tool bar. This is enabled by selecting *Toolbar* under the *View* menu. The tool bar allows faster execution of different menu options by clicking on the different icons. Hovering over the tool bar icons with the mouse causes a tip to pop up that describes the operation of that particular icon.

At the bottom of the Configuration Tool we see the status bar. In Figure 11.3, the status bar shows *Ready* and *No Conflicts*. During manipulation of the configuration, the status bar gives feedback of the operation being performed and any problems that might arise. Additionally, when building the eCos library the status bar gives a progress meter and numerical percentage of the build progress.

During our installation procedure, we selected the build and user tools. However, there might be a circumstance that requires selecting a different location for these tools. In that case, select *Tools -> Paths -> Build Tools* to set the location of the cross-development tools, such as i386-elf-gcc. To set the user tools, select *Tools -> Paths -> User Tools* to set the location of the Cygwin tools.

In certain circumstances, such as trying out a new snapshot of the eCos source code, it might be necessary to have the Configuration Tool use a different local source code repository. The repository the Configuration Tool uses is changed by selecting *Build -> Repository* and then browsing to the location of the local eCos source code. The Configuration Tool then searches for the `ecos.db` file.

#### 11.2.1.1 Saving Configurations

As mentioned earlier in this section, the Configuration Tool uses the title bar to notify you that the configuration currently loaded has been modified but not yet saved. It does this by displaying an asterisk (\*) next to the configuration filename on the title bar.

A configuration can be saved either by using the Save icon on the toolbar or through *File -> Save* or *File -> Save As* on the menu bar. Configuration files have a `.ecc` extension, which stands for eCos Configuration. The saved configuration file contains all of the packages loaded, template used, option settings, and description information, which the Configuration Tool uses to generate the proper eCos image. The configuration files can be viewed using any text editor. A good understanding of configuration files is necessary before attempting to hand edit these files. The Configuration Tool inserts comments into the configuration file to separate different sections of the file.

After saving the configuration file, build and install tree directory structures are created. This is a working directory that the Configuration Tool uses to store files during the build procedure.

The working directory structure created after the configuration file is saved is explained in Chapter 12. Code Listing 11.4 shows a portion of a saved configuration file for the i386 PC target using the Net template.

```

1 cdl_configuration eCos {
2 description "" ;
3
4 # These fields should not be modified.
5 hardware pc ;
6 template net ;
7 package -hardware CYGPKG_HAL_I386 current ;
8 package -hardware CYGPKG_HAL_I386_GENERIC current ;
9 package -hardware CYGPKG_HAL_I386_PC current ;
10 package -hardware CYGPKG_HAL_I386_PCMB current ;
11 .
12 .
13 .
14 package -template CYGPKG_HAL current ;
15 package -template CYGPKG_IO current ;
16 package -template CYGPKG_IO_SERIAL current ;
17 package -template CYGPKG_INFRA current ;
18 package -template CYGPKG_ISOINFRA current ;
19 package -template CYGPKG_KERNEL current ;
20 package -template CYGPKG_ERROR current ;
21 package -template CYGPKG_IO_FILEIO current ;
22 package -template CYGPKG_NET current ;
23 package -template CYGPKG_IO_ETH_DRIVERS current ;
24 };
25 .
26 .
27 .
28 cdl_option CYGNUM_HAL_COMMON_INTERRUPTS_STACK_SIZE {
29 # Flavor: data
30 user_value 4096
31 # value_source user
32 # Default value: CYGPKG_KERNEL ? 4096 : 32768
33 # CYGPKG_KERNEL (unknown) == 0
34 # --> 32768
35 # Legal values: 1024 to 1048576
36 };

```

**Code Listing 11.4** Example of a saved .ecc configuration file for i386 target.

As we see in Code Listing 11.4, the saved configuration file uses the CDL. The main CDL command `configuration` is shown on line 1. This command lists all information about the hardware, template, and packages used for this configuration.

Line 5 shows the `pc` hardware target, and line 6 shows the template used for this configuration, `net`. Lines 7 through 23 list the different packages that are included in this configuration.

On line 28, a different part of the configuration file is shown. Here the CDL command `option` is shown for the `CYGNUM_HAL_COMMON_INTERRUPTS_STACK_SIZE`. The `#` character is used for comments in CDL files. In this case, the comment lines, similar to line 29, give extra information about the CDL command setting. We see on line 30 that the `user_value` parameter is set to 4096. The Configuration Tool uses this value setting during the build procedure.

Configuration files allow you to store the packages and option settings for a specific configuration used to build an eCos image. You are then able to open this configuration file and edit the settings, by changing options or adding packages, to build a new eCos image.

### 11.2.1.2 Importing and Exporting Configurations

The import and export features of the Configuration Tool are useful in saving and restoring partial configurations. The import feature, located under *File* → *Import*, is used to restore the exact configuration from a saved configuration file. An eCos Minimal Configuration, with a `.ecm` ending, includes packages and configuration option value settings for a specific configuration. We use this feature in Chapter 12 to import a RedBoot configuration.

The export feature, located under *File* → *Export*, saves a configuration to a `.ecm` file. This is useful if you need to replicate the exact configuration settings on another host system. Code Listing 11.5 shows a portion of an exported `.ecm` file. Exported minimal configuration files can also be viewed with a text editor.

```
1 cdl_option CYGNUM_HAL_VIRTUAL_VECTOR_CONSOLE_CHANNEL_BAUD {
2 user_value 57600
3 };
4
5 cdl_option CYGNUM_HAL_VIRTUAL_VECTOR_DEBUG_CHANNEL {
6 user_value 1
7 };
```

**Code Listing 11.5** Example of an exported `.ecm` file.

Exported minimal configuration files, which also use the CDL, are similar to the saved configuration files. However, the Configuration Tool does not insert comment lines as it does in the saved configuration files. The exported minimal configuration file contains only the differences from the standard configuration settings.

---

**NOTE** To get all packages and configuration option settings accurately recreated, it is better to use the eCos configuration file (`.ecc`). eCos configuration files contain all information for a particular configuration, whereas the minimal configuration files contain only a subset of the information.



For the example shown in Code Listing 11.5, two configuration options were modified. The first option changed was the baud rate for the console channel (`CYGNUM_HAL_VIRTUAL_VECTOR_CONSOLE_CHANNEL_BAUD`) from the default of 38400 to 57600. This command is stored in the file as shown on lines 1 through 3. The second option changed was the debug channel used (`CYGNUM_HAL_VIRTUAL_VECTOR_DEBUG_CHANNEL`) from 0 to 1. This command is stored as shown on lines 5 through 7.





### 11.2.1.3 Configuration Window

The main window of the Configuration Tool is the *Configuration window*. As we see in Figure 11.3, this window contains a tree-based representation of the configuration items (packages, components, and options) currently loaded into the configuration on the left side of the splitter bar. Each of these configuration items is enclosed in a cell. To the right of the splitter bar is the version of the package—in this case, the version is `current`—or the current option value setting.



Clicking on a particular cell activates that configuration item. Clicking on the right side of the splitter bar allows you to modify a particular option. To end editing of a particular cell, you can either click elsewhere in the Configuration window or press *Enter*. To discard modifications made and revert to the previous value, press the *Escape* key.

The plus and minus sign icons next to each node allow you to expand or contract the packages, components, or options located within that node. Each node contains an icon before the text to assist you in determining the type of module the node represents. The different icons are shown in Table 11.1.

**Table 11.1** Configuration Tool Module Icons

| Icon                                                                                | Description                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Represents a package. Typically located at the highest level and contain components and options for configuration. In Figure 11.3, the packages shown include the <i>eCos HAL</i> , <i>i386 Architecture</i> , and the <i>eCos Kernel</i> . |
|  | Indicates a component. Contain subcomponents or options for configuration. Figure 11.3 includes the <i>HAL Interrupt Handling</i> and <i>ROM Monitor Support</i> components within the <i>eCos HAL</i> package.                             |
|  | Denotes a Boolean flavor option. This icon represents options that can be either enabled or disabled. Figure 11.3 contains the Boolean flavor options <i>Enable I386 FPU Support</i> and <i>Enable Pentium Class CPU Features</i> .         |
|  | Indicates data flavor options of integer type. Options of this type can be set to particular values within their specified ranges.                                                                                                          |

**Table 11.1** Configuration Tool Module Icons (Continued)

| Icon                                                                              | Description                                                                                                                                                                              |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Denotes data flavor options of string type. Options of this type can be set to various value strings.                                                                                    |
|  | Represents data flavor options of enumeration type. Options of this type have predefined values that can be set. The different configuration values are represented in a drop-down menu. |

Icons for particular options are grayed if they are disabled or unable to be modified using the Configuration Tool. Along with the icon being grayed, the option value is also grayed.

Right-clicking on items in the Configuration window brings up a pop-up menu containing:

- **What's This?**—pops up a description of the selected item. This description can also be found in the Short Description window.
- **Properties**—displays a dialog box of the properties for the item. The properties are also found in the Properties window.
- **Restore Defaults**—sets the value of the item to the default value.
- **Visit Documentation**—displays the documentation for the item.
- **View Header File**—opens the header file containing the item. This can also be accomplished by double-clicking the *File* property in the Properties window. The header file is only displayed if the configuration has been saved.
- **Unload Package**—available only when right-clicking on packages. This unloads the selected package from the configuration.

#### 11.2.1.4 Conflicts Window

The *Conflicts window*, shown in Figure 11.3, displays any problems that arise during modification, installation, or removal of configuration items. A conflict occurs when requirements are not met between configuration items that are defined in the CDL.

The Conflicts window is split into three different columns: *Item*, *Conflict*, and *Property*. The *Item* column displays the macro name of the first item involved in the conflict. The *Conflict* column shows a description of the conflict type, such as *Unsatisfied* or *Illegal*. The *Property* column contains a description of the configuration item property that caused the conflict. The Conflicts window can be enabled or disabled by selecting *View -> Conflicts* from the menu bar or using the hot-key combination *Alt+5*.

Right-clicking on a particular item in the Conflicts window pops up a menu with two options, *Locate* and *Resolve*. *Locate* causes the configuration item to be selected in the Configuration window. *Resolve* allows you to resolve the conflict prior to building the eCos library. Resolving conflicts is covered in the *Using Templates* section of this chapter.

### 11.2.1.5 Properties Window

The *Properties window* is located below the Conflicts window, as we see in Figure 11.3. This window displays the CDL properties found in the CDL script file for the currently selected configuration item. A detailed description of the relationship between the CDL script files and the Properties window can be found in the *Graphical Representation of CDL Script Files* section of this chapter. The Properties window can be enabled or disabled by selecting *View -> Properties* from the menu bar or using the hot-key combination *Alt+1*.

Two columns divide the Properties window, *Property* and *Value*. The *Property* column gives the property name found in the CDL script file. Each configuration item contains different properties in the command body. The *Value* column displays the available or current value settings from the CDL script file.

Double-clicking the *URL* property causes the referenced HTML page to be displayed. This is the same as right-clicking the configuration item, in the Configuration window, and selecting *Visit Documentation*. Double-clicking on the *File* property opens the file described in the *Value* column. This file contains the source code for the configuration option settings. The current configuration must be saved for this to work.

### 11.2.1.6 Short Description Window

The *Short Description window* displays a brief description, from the CDL script file, which contains information about the currently selected configuration item. The Short Description window is shown in Figure 11.3.

This window can be enabled or disabled by selecting *View -> Short Description* from the menu bar or using the hot-key combination *Alt+3*. The information displayed in the Short Description window is also displayed when right-clicking the configuration item, in the Configuration window, and selecting *Visit Documentation*.

### 11.2.1.7 Output Window

The *Output window* is located at the bottom of the Configuration Tool as shown in Figure 11.3. The Output window displays messages generated by the execution of external tools. For example, when building the eCos library, the Output window displays commands executed during the build process along with any warning or error messages that occur. The Output window can be enabled or disabled by selecting *View -> Output* from the menu bar or using the hot-key combination *Alt+2*.

Right-clicking in the Output window pops up a menu that allows you to copy text from the window, clear text from the window, select all text in the window, or save the output in the window to a log file.

### 11.2.1.8 Memory Layout Window

As previously mentioned, version 1.3.net of the Configuration Tool includes the Memory Layout Tool, which is not present in version 2.11. Version 1.3.net of the Configuration Tool is located under the `D:\ecos\bin` directory in the file `Configtool113net.exe`, which we installed

in Chapter 10. Version 1.3.net is used to graphically manipulate memory configurations. The other alternative is to edit the memory configuration files by hand.

The *mEmory Layout window* is a graphical view of the memory configuration based on the currently selected hardware architecture, platform, and Startup Type configuration option. A horizontal bar that is divided into a number of blocks for each memory section represents the memory layout for a region. The name of the memory region is above the horizontal bar along with the address range covered by that region. For example, in Figure 11.4 we see the MLT's graphical representation of the memory configuration for the Motorola PowerPC MBX860 platform with a ROM Startup Type.

The Memory Layout window can be enabled or disabled by selecting *View -> Memory Layout* from the menu bar or using the hot-key combination *Alt+4*. In version 1.3.net of the Configuration Tool, the memory layout tool bar can be used to modify memory configurations. This tool bar can be enabled or disabled by selecting *View -> Toolbars -> Memory Layout* from the menu bar. Right-clicking in the Memory Layout window allows you to view the properties of the selected region or section of memory.

#### 11.2.1.9 Memory Layout Manipulation

There are default memory layouts provided for all supported platforms that do not need to be modified to begin developing using eCos. Modifying the memory layout is necessary when porting eCos to your own platform, which probably differs from any supported platform, or if memory is added to or removed from an evaluation board. Modifications can be made in the Memory Layout window or using the memory layout tool bar.

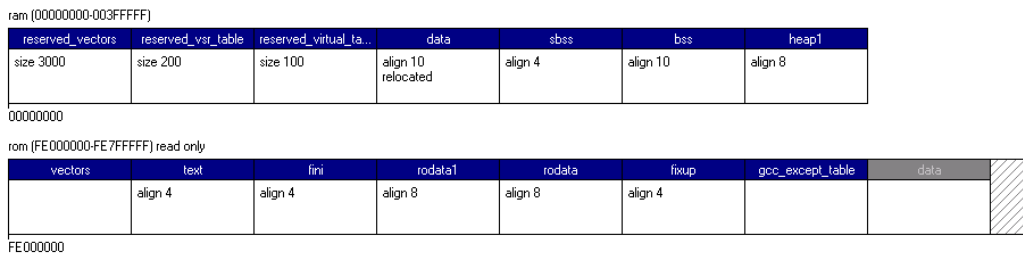
The memory layout files for each platform are found under the `include\pkgconf` sub-directory within the HAL platform directory structure. Typically, each platform includes RAM and ROM memory layout files that are used by the Configuration Tool based on the Startup Type configuration option selection. Some platforms also contain ROMRAM memory layout files for applications that start in ROM but are copied to RAM for execution. For each Startup Type supported by the platform, there are three different memory layout files.

The first type of memory layout file contains a `.h` extension, which contains C macro definitions of the memory region. The `.ldi` files are linker script files that define region and section locations. Finally, the `.mlt` files contain the description of the memory layout for use by the MLT. When editing the memory layout files by hand, only the `.h` and `.ldi` files need to be modified; the `.mlt` file is only used by the MLT.

Figure 11.4 shows the Memory Layout window for the Motorola PowerPC MBX860 platform with a ROM Startup Type configuration option selected. There are two region names in this memory layout, `ram` and `rom`. We can see that the `ram` region includes the address range `0x0000_0000` to `0x003F_FFFF`, and the `rom` region includes the address range `0xFE00_0000` to `0xFE7F_FFFF`.

The different memory sections, such as `reserved_vectors` in the `ram` region and `text` in the `rom` section, are shown in Figure 11.4. Within each memory section is a short description about that section, which might include the size, alignment, and whether the section

is relocated. We can see in Figure 11.4 that the `reserved_vectors` section in the `ram` region has a size of 3000 bytes, while the `text` section in the `rom` region is aligned on a 4-byte boundary. The base addresses for the two memory regions start at the left with the lowest addresses for that region shown below the horizontal bar. Unused portions of memory are represented with hatching, as shown at the end of the `rom` region in Figure 11.4.



**Figure 11.4** Memory Layout window configuration for Motorola PowerPC MBX860 platform with ROM Startup Type.

Code Listing 11.6 contains a fragment of the linker script file, `mlt_powerpc_mbx_rom.ldi`, for the Motorola PowerPC MBX860 platform with ROM Startup Type. This `.ldi` file is generated based on the memory layout configured in Figure 11.4. For additional information about linker scripts and linker commands, see the GNU `ld` documentation, which can be found online at:

[www.gnu.org/manual/manual.html](http://www.gnu.org/manual/manual.html)

```

1 MEMORY
2 {
3 ram : ORIGIN = 0, LENGTH = 0x400000
4 rom : ORIGIN = 0xfe000000, LENGTH = 0x800000
5 }
6
7 SECTIONS
8 {
9 SECTIONS_BEGIN
10 SECTION_vectors (rom, 0xfe000000, LMA_EQ_VMA)
11 SECTION_text (rom, ALIGN (0x4), LMA_EQ_VMA)
12 SECTION_fini (rom, ALIGN (0x4), LMA_EQ_VMA)
13 .
14 .
15 .
16 SECTIONS_END
17 }
```

**Code Listing 11.6** Linker script fragment for Motorola PowerPC MBX860 platform with ROM Startup Type.

The first command, `MEMORY`, on line 1 defines the memory regions present in the layout. In Code Listing 11.6, we can see that the `ram` and `rom` region definitions are on lines 3 and 4, respectively. The `ram` region starts at address 0, denoted by the `ORIGIN = 0` command on line 3, and has a length of `0x0040_0000`, also shown on line 3. The `rom` region contains the same commands with different start address and length values as we can see on line 4. These correspond to the graphical representation for the `ram` and `rom` regions shown in Figure 11.4.

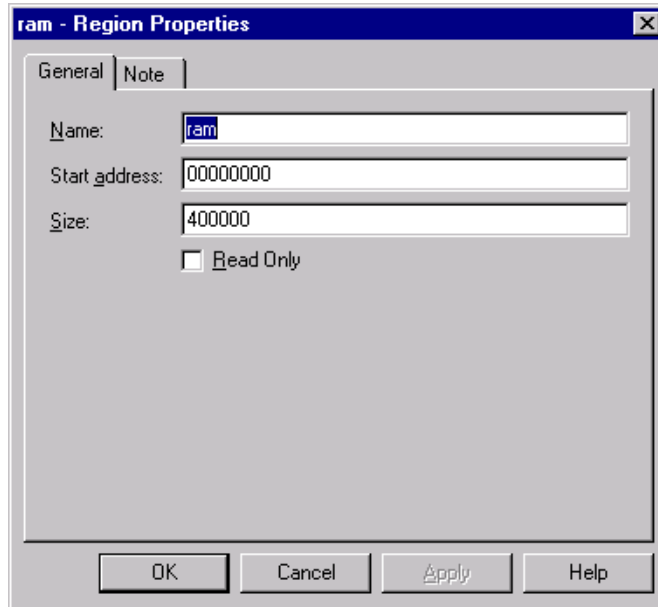
The `SECTIONS` command defines the different sections present in the memory layout. The macro `SECTION_XXX`, where `XXX` defines the section, takes the parameters `region`, `VMA`, and `LMA`. On line 10, for example, we see that the `vectors` section has a final location at the beginning of the `rom` region at absolute address `0xFE00_0000`. The next section, `text`, is also located in the `rom` section and follows the `vectors` section, since the `VMA` address is specified as `ALIGN (0x4)`. The symbol `LMA_EQ_VMA`, or `LMA` equals `VMA`, means that the section is not relocated. We can see the graphical representation of the `vectors` and `text` sections in the `rom` region in Figure 11.4.

Modifying or creating memory regions is accomplished using a property sheet. The Property Sheet dialog box for a memory region is displayed either by selecting the *Properties* icon on the memory layout tool bar, after the specific region is selected, or double-clicking the desired region. To select a memory region, you can click on the region name. The `ram` region Property Sheet dialog box is shown in Figure 11.5. New memory regions can be created using the *New Region* icon on the memory layout tool bar.

The *General* tab is displayed in Figure 11.5. The information in this dialog box ensures that initial and final locations of relocated memory sections are within the appropriate memory region. The *Name* chosen for a memory region is up to you; however, it should not contain spaces or punctuation characters. In Figure 11.5, the name of the region is `ram`. The *Start Address* and *Size* are specified in bytes and entered as hexadecimal numbers. We can see in Figure 11.5 that the *Start Address* for the `ram` region is `0x0000_0000` and the *Size* is `0x0040_0000`, which corresponds to a size of 4 Mbytes (4,194,304 bytes). Therefore, the address range for the `ram` region, as we see in Figure 11.4, is from `0x0000_0000` to `0x003F_FFFF`. In this case, the *Read Only* check box is not checked because this is a RAM region of memory. The *Notes* tab can be used to keep any information about the memory region.

Memory sections are also edited using a property sheet. The Property Sheet dialog box for a memory section is displayed either by selecting the *Properties* icon on the memory layout tool bar, after selecting a specific region, or double-clicking the desired section. New memory sections can be created using the *New Section* icon on the memory layout tool bar. Figure 11.6 shows the *General* tab of the `reserved_vsr_table` section from the Motorola PowerPC MBX860 platform memory layout.

A memory section is either *Linker-defined* or *User-defined*. In Figure 11.6, we see this section is *User-defined* with the name `reserved_vsr_table`. The same rules apply for naming memory sections as with memory regions—no spaces or punctuation characters. If the section is *Linker-defined*, a drop-down list containing the names currently not used are presented. The names in the list vary depending on the selected target architecture. When using a template as

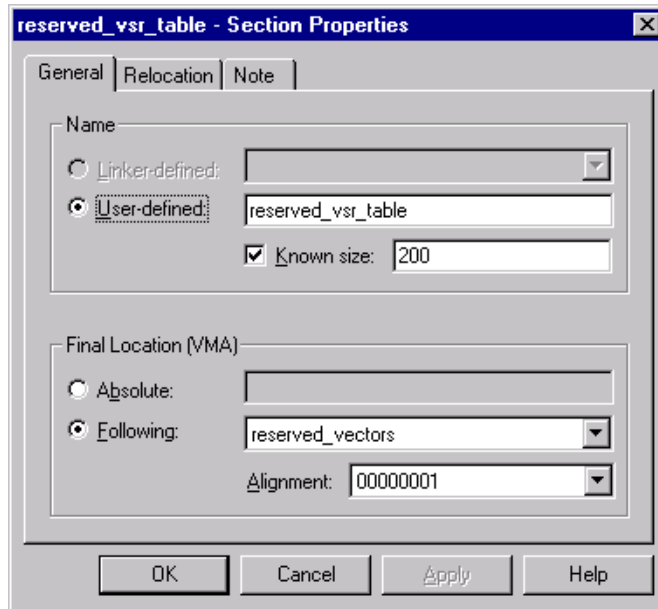


**Figure 11.5** ram region property sheet dialog box.

the baseline for your development, sections will be set up in the memory layout for you. It is then up to you to determine if they need to be relocated or moved into other memory addresses or if new sections need to be added.

If the *Known Size* check box is checked, then the size, in bytes, is entered as a hexadecimal number. In Figure 11.6, we see that the `reserved_vsr_table` has a size of 200, or 512 bytes. *User-defined* sections that do not contain a size are assumed to occupy all memory up to the next section or the end of memory in that region. The *Final Location*, or *Virtual Memory Address* (VMA), defines the final location of the section after relocation. Either an *Absolute* address can be entered or *Following* a specific section. In Figure 11.6, the `reserved_vsr_table` follows the `reserved_vectors` section, which was selected from a drop-down list. The *Alignment* of the `reserved_vsr_table` is also specified on a 1-byte boundary. If an *Absolute* address is used, this should be entered in hexadecimal.

The next tab in the section property sheet is for *Relocation*. For the Motorola PowerPC MBX860 platform with ROM Startup Type, the only section that is relocated is the data section. We can see this in Figure 11.4 in the ram region that has the description `relocated` in the data section box below `align 10`. In Figure 11.7 we see the *Relocation* tab for the data section. The *Relocate Section* check box is checked and the *Initial Location*, or *Load Memory Address* (LMA), for the data section is specified. The LMA can be either an *Absolute* address in hexadecimal or *Following* a specific section that is selected from a drop-down list. In Figure 11.7, the data section LMA is initially loaded in the rom region following the `gcc_except_table` section. The *General* tab in the data section property sheet specifies a



**Figure 11.6** reserved\_vsr\_table section Property Sheet dialog box.

VMA, or final location, following the reserved\_virtual\_table section. If we look at Figure 11.4 we can see that the data section is present in both the rom and ram region. Since the data section is relocated from the rom to ram region, it is grayed in the rom region.

The last tab is the *Notes* tab, which can be used to keep any information about the memory section.

Let's briefly look at how we would access a user-defined memory section. In this example, we create a user-defined section using the MLT and then go through a code listing showing how this section is accessed. We are using the Motorola PowerPC MBX860 platform with ROM Startup Type.

First, the section kfm1 is created using the MLT, as shown in Figure 11.8. We see the MLT representation of the RAM and ROM memory regions. In the RAM memory region, we see the kfm1 section at the end. The section Properties dialog box is also shown for the kfm1 section. In this case, the section starts at address 0x000A\_0000 and is 296 bytes long.

When we save our configuration, the Configuration Tool generates the memory layout header file, mlt\_powerpc\_mbx\_rom.h, automatically. This file is stored in the install tree directory structure under the include\pkgconf directory. An excerpt from the memory layout header file with the kfm1 section defined is shown in Code Listing 11.7.

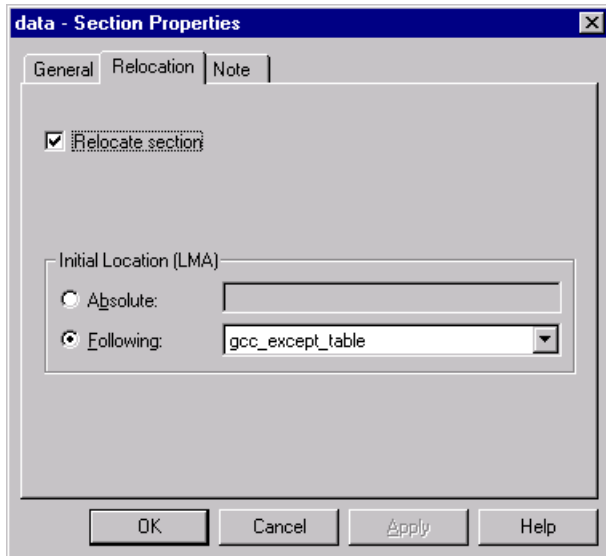
```

1 #define CYGMEM_SECTION_kfm1 (CYG_LABEL_NAME (__kfm1))
2 #define CYGMEM_SECTION_kfm1_SIZE (0x128)

```

**Code Listing 11.7** Memory layout file excerpt showing kfm1 section definitions.





**Figure 11.7** data section property sheet dialog box.

In Code Listing 11.7, we see that line 1 shows the address definition of our `kfm1` section we defined using the MLT. The actual address, `0x000A_0000`, is contained in the linker script file `mlt_powerpc_mbx_rom.ldi`, which is also generated by the Configuration Tool. On Line 2, the size for our new section is defined; in this case, the size is 296 (128 in hexadecimal) bytes.

Code Listing 11.8 shows the example code we implement in our application to access our user-defined section `kfm1`.

```

1 #include <pkgconf/system.h>
2 #include <pkgconf/mlt_powerpc_mbx_rom.h>
3
4 //
5 // Main starting point for the application.
6 //
7 void cyg_user_start(void)
8 {
9 // Use the memory section as an integer array.
10 char *user_section = (char *)CYGMEM_SECTION_kfm1;
11 unsigned char user_sect_size = CYGMEM_SECTION_kfm1_SIZE;
12 unsigned int index;
13
14 // Initialize each element in the kfm1 section of memory.
15 for (index = 0; index < user_sect_size; index++)
16 user_section[index] = 0;
17 }

```

**Code Listing 11.8** Example of how to access a user-defined memory section.



In Code Listing 11.8, we see that the header file, `mlt_powerpc_mbx_rom.h`, which defines our new user section `kfm1` is included on line 2. On line 10 we set the variable `user_section` to our user-defined section address, which allows us to access the section using this local variable. The size of the section is set to the variable `user_sect_size` on line 11, again allowing us to use a local variable to access the size of our user-defined section. Finally, we initialize the user-defined memory section `kfm1` to 0, which is shown on lines 15 and 16, using our local variables `user_section` and `user_sect_size`.

### 11.2.2 eCos Repository Database

The Configuration Tool uses the eCos repository database to understand the components within the repository. The descriptions of the packages in the repository database are contained in the file `ecos.db`, which is located under the `D:\ecos\packages` subdirectory in our installation. This file uses the CDL to describe all packages and targets in the database.

The Configuration Tool needs to know the location of this file, and the entire repository. The Configuration Tool searches for the component repository by first using the most recently used repository location, then by using the default location setup by the installation, and finally, by whatever path you select. The component repository location should already be configured properly from our installation.

If you needed to change the location of the component repository, you could do so by selecting *Build -> Repository* from the menu bar. This brings up a dialog box that allows you to browse to the new location of the repository.

We can see an example of the relationship between the repository database file, `ecos.db`, and the Configuration Tool in Figure 11.9. In Figure 11.9, the CDL package description for the eCos HAL is shown along with the Configuration window, the Properties window, and the Short Description window from the Configuration Tool. We can see how the Configuration Tool interprets the CDL package for the eCos HAL in order to display the properties and description correctly. The Configuration Tool uses the CDL script file, `hal.cdl`, shown on line 4 of the `ecos.db` code fragment in Figure 11.9, to display the proper components within the eCos HAL package.

Along with all of the package descriptions in the `ecos.db` file, the template descriptions are also contained within the database file. Template descriptions are used by the Configuration Tool to load pre-configured packages for a particular target platform. Details about using templates can be found in the *Using Templates* section of this chapter. The template description for the i386 PC target platform is shown in Code Listing 11.9.

We see in Code Listing 11.9, line 1 contains the name of the target; in this case, `pc`. An alias for the target, which is displayed by the Configuration Tool when selecting a template, is shown on line 2. The packages that are loaded by the Configuration Tool when the i386 PC target template is selected are given on lines 3 through 13. These packages give a baseline of functionality for a given target. Packages can then be configured, added, or removed in order to get the specific configuration you need for your target platform. Finally, lines 15 through 19 give the description of this template.

```

1 package CYGPKG_HAL {
2 alias { "eCos common HAL" hal hal_common }
3 directory hal/common
4 script hal.cdl
5 description "
6 The eCos HAL package provide a porting layer for
7 higher-level parts of the system such as the kernel
8 and the C library. Each installation should have
9 HAL packages for one or more architectures, and
10 for each architecture there may be one or more
11 supported platforms. It is necessary to select one
12 target architecture and one platform for that
13 architecture. There are also a number of configuration
14 options that are common to all HAL packages."
15 }

```

| Property     | Value                                          |
|--------------|------------------------------------------------|
| URL          | redirect:/the-ecos-hardware-abbreviation-layer |
| Macro        | CYGPKG_HAL                                     |
| File         | include\pkgconf\hal.h                          |
| Value        | current                                        |
| Default      | current                                        |
| Type         | String                                         |
| IncludedDir  | cyg/hal                                        |
| Doc          | redirect:/the-ecos-hardware-abbreviation-layer |
| Compile      | generic-stub.c thread-packets.c hal_stub.c c   |
| Compile      | -libgcc=libgcc.a dummy.c                       |
| Make         | -priority 250 <PREFIX>/lib/exit.as.o: <PREFIX> |
| Package-Data |                                                |

The eCos HAL package provide a porting layer for higher-level parts of the system such as the kernel and the C library. Each installation should have HAL packages for one or more architectures, and for each architecture there may be one or more supported platforms. It is necessary to select one target architecture and one platform for that architecture. There are also a number of configuration options that are common to all HAL packages.

Figure 11.9 eCos repository database, ecos.db, and Configuration Tool relationship.

```
1 target pc {
2 alias { "i386 PC target" }
3 packages { CYGPKG_HAL_I386
4 CYGPKG_HAL_I386_GENERIC
5 CYGPKG_HAL_I386_PC
6 CYGPKG_HAL_I386_PCMB
7 CYGPKG_IO_PCI
8 CYGPKG_IO_SERIAL_GENERIC_16X5X
9 CYGPKG_IO_SERIAL_I386_PC
10 CYGPKG_DEVS_ETH_INTEL_I82559
11 CYGPKG_DEVS_ETH_I386_PC_I82559
12 CYGPKG_DEVICES_WALLCLOCK_DALLAS_DS12887
13 CYGPKG_DEVICES_WALLCLOCK_I386_PC
14 }
15 description "
16 The pc target provides the
17 packages needed to run eCos
18 binaries on a standard i386
19 PC motherboard."
20 }
```

**Code Listing 11.9** i386 PC target template description from eCos database file `ecos.db`.

Figure 11.10 shows the template dialog box displaying the i386 PC target from Code Listing 11.9. The template dialog box is displayed when *Build -> Templates* is selected from the menu bar. We can see in Figure 11.10 that the alias description, from line 2, is displayed in the Hardware drop-down list.

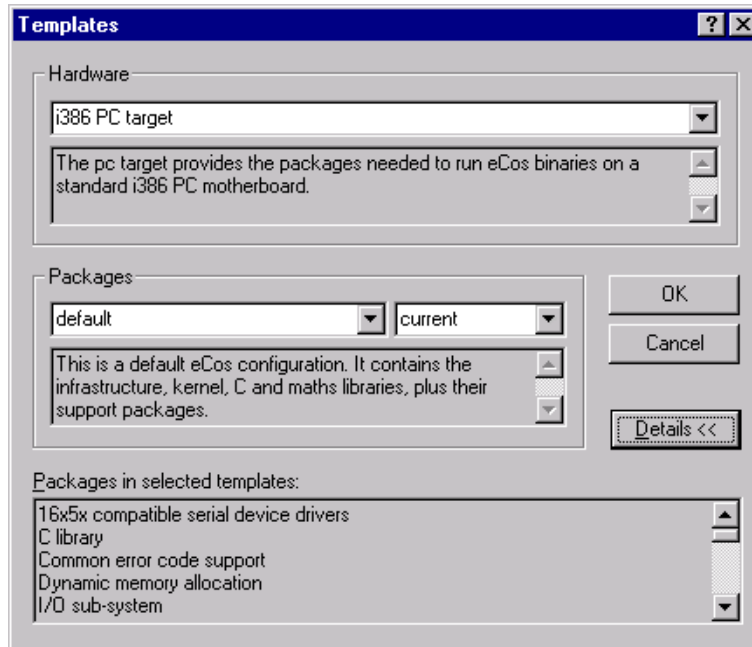
Below the target alias is the description for the template, from lines 15 through 19. In this case, the `default` packages are included, which are described in the *Packages in Selected Templates* description box. These packages are listed on lines 3 through 13 in Code Listing 11.9. The different options in the package drop-down list are described in the *Using Templates* section of this chapter.

In certain circumstances, it might be necessary to edit the `ecos.db` file in order to have the Configuration Tool recognize a package you added. We look at editing the `ecos.db` file in Chapter 13, *Porting eCos*.

A file named `ChangeLog` is also located in the same subdirectory as the `ecos.db` file. This log file tracks the changes made to the `ecos.db` file. Entries to this log file are similar to the entries in the other `ChangeLog` files described in the *Package Directory Structure* section in this chapter.

### 11.2.3 Graphical Representation of CDL Script Files

Now that we have looked at the relationship between the repository database file and the Configuration Tool, let's go a step further and examine the relationship between the CDL script files and their graphical representation in the Configuration Tool.



**Figure 11.10** Configuration Tool template selection dialog box.

This section gives us a basic understanding for how the Configuration Tool interprets the commands in the CDL script files and uses this information in its different display windows. All CDL commands and properties are not described in this section; however, you should be able to see the basic relationship between the CDL scripts and the Configuration Tool allowing you to investigate other script details on your own.

As we know from the *Packages* section in this chapter, every package must have at least one CDL script file to define the package. The Configuration Tool uses the CDL script files to display the necessary configuration and description information so that you can set up the package according to your needs. We use the same CDL script file from the i386 PC HAL package described in Code Listing 11.3, which you can look at to get an overview of where each portion of the CDL script commands are located. The line numbers from Code Listing 11.3 are used in the code fragments located in the figures. Figures 11.11, 11.12, and 11.13 show three different portions of the CDL script file and how the Configuration Tool displays this information. Each figure shows a portion of the Configuration window, the Properties window, and the Short Description window.

First, in Figure 11.11, is the CDL package command. Line 1 gives the macro name of the package `CYGPKG_HAL_I386_PC`, which can also be seen next to the `Macro` property in the Properties window. Line 2 uses the `display` command to give a description of the package, which we can see is used in the Configuration window. The i386 PC Target is part of the i386 Architecture. In order to locate the i386 PC Target package under the i386 Architecture, the `parent` command shown on line 3 is used.

```

1 cdl_package CYGPKG_HAL_I386_PC {
2 display "i386 PC Target"
3 parent CYGPKG_HAL_I386

```

```

4 define_header hal_i386_pc.h
5 include_dir cyg/hal
6
7
8
9
10
11 compile hal_diag.c plf_misc.c plf_stub.c
12
13 implements CYGINT_HAL_DEBUG_GDB_STUBS
14 implements CYGINT_HAL_DEBUG_GDB_STUBS_BREAK
15 implements CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT

```

```

1 cdl_package CYGPKG_HAL_I386_PC {
2 display "i386 PC Target"
3 parent CYGPKG_HAL_I386

```

| Property     | Value                                                |
|--------------|------------------------------------------------------|
| URL          | reflect/the-ecos-hardware-abstraction-layer/hal.html |
| File         | D:\Temp\vxw\ecos\i386net_install\include\pkgcom...   |
| Macro        | CYGPKG_HAL_I386_PC                                   |
| Value        | current                                              |
| Default      | current                                              |
| Type         | String                                               |
| Parent       | CYGPKG_HAL_I386                                      |
| DefineHeader | hal_i386_pc.h                                        |
| IncludeDir   | cyg/hal                                              |
| Compile      | hal_diag.c plf_misc.c plf_stub.c                     |
| Implements   | CYGINT_HAL_DEBUG_GDB_STUBS                           |
| Implements   | CYGINT_HAL_DEBUG_GDB_STUBS_BREAK                     |
| Implements   | CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT                    |
| DefineProc   | CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT_...                |

```

6 description "
7 The i386 PC Target HAL package provides
8 the support needed to run eCos binaries
9 on an i386 PC."

```

|              |
|--------------|
| current      |
| src/i386.id  |
| current      |
| current      |
| RAM          |
| 38400        |
| 38400        |
| 3            |
| 1            |
| 0            |
| 0            |
| i386_pc_rair |
| current      |
| current      |

```

6 description "
7 The i386 PC Target HAL package provides
8 the support needed to run eCos binaries
9 on an i386 PC."

```

| Property     | Value                                                |
|--------------|------------------------------------------------------|
| URL          | reflect/the-ecos-hardware-abstraction-layer/hal.html |
| File         | D:\Temp\vxw\ecos\i386net_install\include\pkgcom...   |
| Macro        | CYGPKG_HAL_I386_PC                                   |
| Value        | current                                              |
| Default      | current                                              |
| Type         | String                                               |
| Parent       | CYGPKG_HAL_I386                                      |
| DefineHeader | hal_i386_pc.h                                        |
| IncludeDir   | cyg/hal                                              |
| Compile      | hal_diag.c plf_misc.c plf_stub.c                     |
| Implements   | CYGINT_HAL_DEBUG_GDB_STUBS                           |
| Implements   | CYGINT_HAL_DEBUG_GDB_STUBS_BREAK                     |
| Implements   | CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT                    |
| DefineProc   | CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT_...                |

Figure 11.11 CDL script file, showing CDL package command, and Configuration Tool representation for the i386 PC HAL package.

```

17 cd1_component CYG_HAL_STARTUP {
18 display
19 flavor data
20 legal_values {"RAM" "FLOPPY" "ROM"}
21 default_value {"RAM"}
22 no_define
23 define
24 -file system.h CYG_HAL_STARTUP

```

The screenshot shows the Configuration Tool interface. On the left, a tree view shows the component hierarchy: **cd1\_component** (expanded) → **display** (expanded) → **Start-up type** (expanded). The **Start-up type** property is selected, and its value is set to **FLOPPY**. Below the tree, a table lists various properties and their values:

| Property      | Value                                                  |
|---------------|--------------------------------------------------------|
| URL           | redirect:/the-ecos-hardware-abstraction-layer-hal.html |
| File          | D:\Temp\www\ecos386net_install\include\pkgonc\k        |
| Macro         | CYG_HAL_STARTUP                                        |
| Value         | RAM                                                    |
| Default       | RAM                                                    |
| Type          | Enumeration                                            |
| Flavor        | data                                                   |
| Default/Value | "RAM"                                                  |
| Legal/Values  | "RAM" "FLOPPY" "ROM"                                   |
| No/Define     |                                                        |
| Define        | -file system.h CYG_HAL_STARTUP                         |

On the right side of the tool, a text box provides additional information:

It is possible to configure eCos for the PC target to build for RAM startup (RedBoot), FLOPPY startup (for writing to a floppy disk), or ROM startup (for writing straight to a boot ROM/Flash). ROM startup is experimental at this time.

Below the screenshot, a CDL script file is shown with the following content:

```

19 flavor data
20 legal_values {"RAM" "FLOPPY" "ROM"}
21 default_value {"RAM"}
22 no_define
23 define
24 -file system.h CYG_HAL_STARTUP

```

At the bottom of the screenshot, a description is provided:

```

24 description "
25 It is possible to configure eCos
26 for the PC target to build for RAM
27 startup (generally when being run
28 under an existing monitor program
29 like RedBoot), FLOPPY startup (for
30 writing to a floppy disk, which can
31 then be used for booting on PCs with
32 a standard BIOS), or ROM startup
33 (for writing straight to a boot
34 ROM/Flash). ROM startup is experimental
35 at this time."

```

Figure 11.12 CDL script file, showing CDL component command, and Configuration Tool representation for the i386 PC HAL package.



The different properties, on lines 4, 5, and 11 through 15, for the i386 PC Target are displayed in the Properties window. The `description` command from lines 6 through 9 is shown in the Short Description window.

Within the i386 PC Target package is the Startup Type component. The CDL script file and Configuration Tool representation for the Startup Type component is shown in Figure 11.12. Because this component is part of the CDL package command `i386 PC Target (CYG_HAL_I386_PC)` body, the Startup Type component is displayed under the i386 PC Target in the Configuration window hierarchy structure.

Again, we see the macro name, `CYG_HAL_STARTUP`, on line 17, which is also displayed in the Properties window. The `display` command, on line 18, gives the Configuration Tool a descriptive name for the component to display in the Configuration window.

The properties on lines 19 through 23 are displayed in the Properties window. One note is that since this component has a `flavor` property of `data` and a `type` property of `enumeration`, a drop-down list is used to show the `legal_values` to configure this component. We can see the `enumeration` type icon, as previously described in Table 11.1, next to the Startup Type description in the Configuration window. The `legal_values` for the i386 PC Target package Startup Type are `RAM`, `FLOPPY`, and `ROM`.

Finally, we see the `description` command from lines 24 through 35 displayed in the Short Description window.

Finally, we look at the CDL option command `Output to PC Screen` in Figure 11.13. The macro for this option is on line 38, `CYGSEM_HAL_I386_PC_DIAG_SCREEN`. This option is also located within the i386 PC Target package as we can see from the hierarchy structure shown in the Configuration window. Line 39 shows the `display` command represented in the Configuration window with the text `Output to PC Screen`.

A check box icon to the left of the display text is used for this option since the `flavor` property is `bool`, as shown on line 40 and in the Properties window. The other properties on lines 41 and 42 are displayed in the Properties window as well.

As we have seen with the other CDL script fragments, the `description` command is used by the Configuration Tool to display text in the Short Description window, shown on lines 43 through 46.

### 11.2.4 Using Templates

*Templates* are predefined configurations provided in the eCos repository that load specific packages for a supported hardware platform. As described in Chapter 1, templates are used as a baseline starting point to begin selecting the configuration settings specific to your application.

The templates dialog box, shown in Figure 11.10, is displayed by selecting *Build* → *Templates* from the menu bar. We use a template in Chapter 12 to load a baseline of packages and configuration options for our example application.

```

38 cd1_option CYGSEM_HAL_I386_PC_DIAG_SCREEN {
39 display "Output to PC screen"

```

```

40 flavor bool
41 default_value 1
42 implements CYGINT_HAL_I386_PCMB_SCREEN_SUPPORT

```

| Property      | Value                                                  |
|---------------|--------------------------------------------------------|
| URL           | redirect:/the-ecos-hardware-abstraction-layer-hal.html |
| File          | D:\T\emp\xxx\ecos\i386\net_install\include\pkgconf\N   |
| Macro         | CYGSEM_HAL_I386_PC_DIAG_SCREEN                         |
| Enabled       | True                                                   |
| Flavor        | bool                                                   |
| Default Value | 1                                                      |
| Implements    | CYGINT_HAL_I386_PCMB_SCREEN_SUPPORT                    |

```

43 description "
44 This option enables use of the PC screen
45 and keyboard as a third virtual serial
46 device."

```

Figure 11.13 CDL script file, showing CDL option command, and Configuration Tool representation for the i386 PC HAL package.

---

**NOTE** Prior to using templates for different hardware architectures you must make sure that the platform-specific cross-development tools have been built for the hardware architecture you have selected. You must also make sure that the appropriate cross development tools are selected from the *Tools -> Path -> Build Tools* menu in the Configuration Tool.

The Configuration Tool will display errors if you try to configure and build the eCos library with the wrong cross-development tools installed. The installation of the cross-development tools is covered in Chapter 10.

The templates dialog box allows the selection of a combination of hardware and package templates. The hardware templates are defined in the eCos database file, `ecos.db`, using CDL script commands. The hardware template is selected from the drop-down list in the Templates dialog box under the *Hardware* label.

Beneath the drop-down list is a brief description of the target platform selected. The hardware template ensures that the proper packages are loaded into the Configuration Tool for a particular target platform, including HAL, I/O, and device driver packages. There is a hardware template in the drop-down list for each of the supported evaluation platforms supported by eCos, as shown in Appendix A, *Supported Processors and Evaluation Platforms*.

The package templates are generic across all hardware platforms and are defined under the `templates` subdirectory located under the `D:\ecos\packages` directory. The package template is also selected from a drop-down list under the *Packages* label in the Templates dialog box.

Next to the package template list is a drop-down list that allows you to select the version of the package template to load. In this case, the `current` template version is selected.

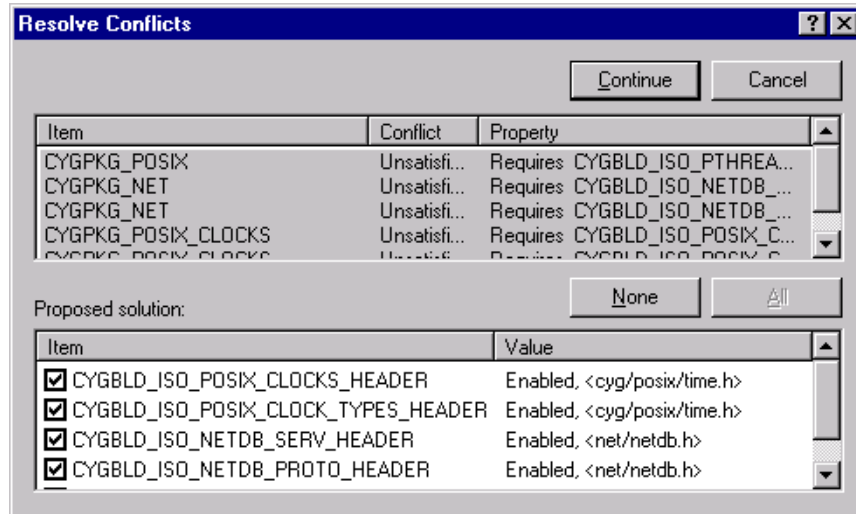
A brief description of the package template selected is located beneath the drop-down lists. Clicking the *Details* button displays a scroll list of the packages that are contained in the package template selected, as shown in Figure 11.10.

Each package template contains its own subdirectory containing a file named `current.ect`, as well as a `ChangeLog` file. You can generate your own package template files, which can be useful if you need to incorporate different versions of various packages. Before the new package template is noticed, the Configuration Tool needs to be restarted. The package template files use CDL scripts to define the proper packages to load and configuration options to set according to the template. The package templates included with eCos are defined in Table 1.2 in Chapter 1.

#### 11.2.4.1 Conflicts and Resolutions

When templates are loaded into the Configuration Tool, oftentimes conflicts arise between different components and configuration options. How and when conflicts are displayed depends on the settings selected under the *Tools -> Options* menu. This allows you to check for conflicts after any item is changed, before saving and building, or never.

When the conflict option *Automatically Suggest Fix* is selected, a dialog box is displayed when a conflict arises by the Configuration Tool, similar to the one shown in Figure 11.14, which allows you to resolve the conflicts before proceeding with the configuration setup.



**Figure 11.14** Resolve Conflicts dialog box.

As we see in Figure 11.14, the top window displays the conflict that needs to be addressed with the columns Item, Conflict, and Property, which is the same format as the Conflicts window. The bottom window gives a *Proposed Solution* for each conflict. Each proposed solution can be individually enabled or disabled by selecting the check box in the Item column or a global enable, provided by the *All* button, and disable, provided by the *None* button, are included in the dialog box. The default state is that all proposed solutions are enabled. Clicking the *Continue* button applies the proposed solutions selected. Clicking the *Cancel* button exits the dialog box without applying any solutions.

If you choose not to apply the proposed solutions, the resolve conflicts dialog box can be displayed later by selecting *Tools -> Resolve Conflicts* from the menu bar. Unresolved conflicts are also displayed in the Conflicts window and a total count is shown on the bottom-right side of the status bar.

---

**NOTE** There are a couple of points to understand about conflicts and resolutions. First, an automatic resolution might not be generated for every conflict that comes about. You might need to resolve the conflict yourself by altering the configuration in some way. If this is the case, it is useful to use the Find tool, as described in *The Configuration Tool* section of this chapter, to search for the macro causing the conflict. The property column in the Conflicts window gives you a hint as to what the problem is and how you can resolve it.

Finally, resolving a conflict might cause other conflicts to occur. In this case, you might, again, need to search and edit the configuration to resolve the conflict. In other circumstances, you might need to unload or load particular packages to resolve the conflict.

### 11.2.5 Package Control

After a template has been loaded, it is often necessary to load or unload particular packages to set up the configuration to support all of the features for your application. Adding and removing packages is accomplished by using the Packages dialog box as shown in Figure 11.15. This dialog box is displayed by selecting *Build* → *Packages* from the menu bar.

The left side of the Packages dialog box contains a scroll list of all available packages in the eCos component repository under the label *Available Packages*. The right side contains a scroll list of the packages currently selected in the configuration under the label *Use These Packages*. The version of the currently selected package is shown in the drop-down list below the label *Version*. If multiple packages are selected, the versions common to both packages are displayed in the version drop-down list. A brief description of the package is located below the version. The description window is blank when multiple packages are selected. In Figure 11.15, we see that the *current* version of the *RAM Filesystem* package is selected from the available packages.

To add a package to the configuration, select the package from the available packages list and click the *Add* button. Multiple packages can be selected by holding down the *Ctrl* key while clicking packages. Next, you select the version of the package you want to add from the *Version* drop-down list. This contains all versions installed under your component repository installation. Since the *current* version is the one we installed in Chapter 10, this is the only version shown.

To remove a package from the configuration, select the package from the current packages being used and then click the *Remove* button. Again, multiple packages can be selected by holding down the *Ctrl* key while clicking packages.

---

**NOTE** An error message might be displayed when you try to add particular packages from the package control dialog box. For example, if you were to try to add a device driver that is not supported by your selected platform, a message box appears informing you that you need to load a different hardware template in order to load the package you just specified. This means that there is some conflict between the packages you have currently selected and the package you are trying to add. In this case, the hardware template selected does not support the device driver package you are trying to add.

### 11.3 Other eCos Tools

There are two additional tools installed with the eCos development kit that we cover in this section. The first is the Package Administration tool. The Package Administration tool allows you to control the packages contained in your local eCos component repository.

The other tool, the command-line configuration tool, allows you to configure packages the same way as the graphical Configuration Tool. This section is intended to give you an overview

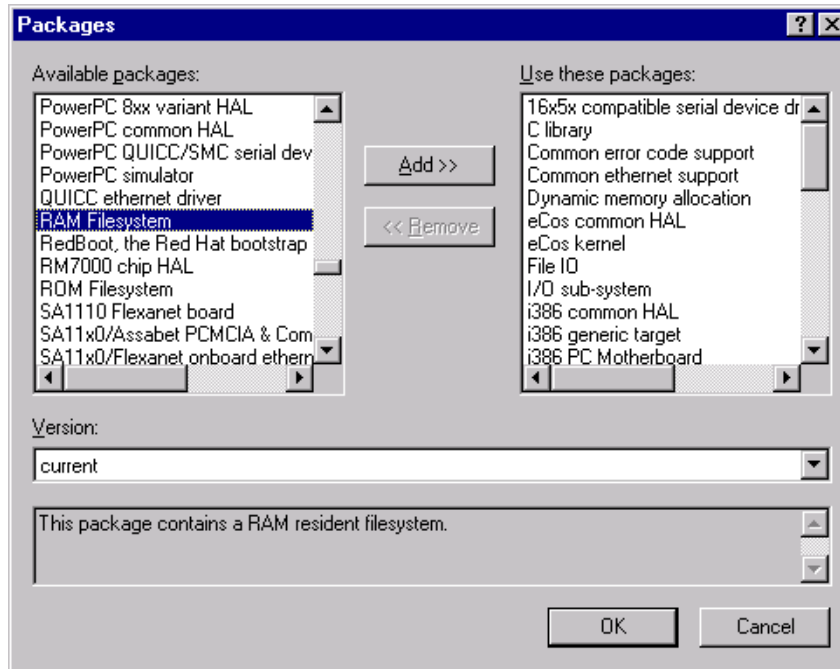


Figure 11.15 Package control dialog box.

of the tools and a basic understanding of how they operate. Additional information for these tools can be found online at:

<http://sources.redhat.com/ecos/docs.html>

### 11.3.1 The Package Administration Tool

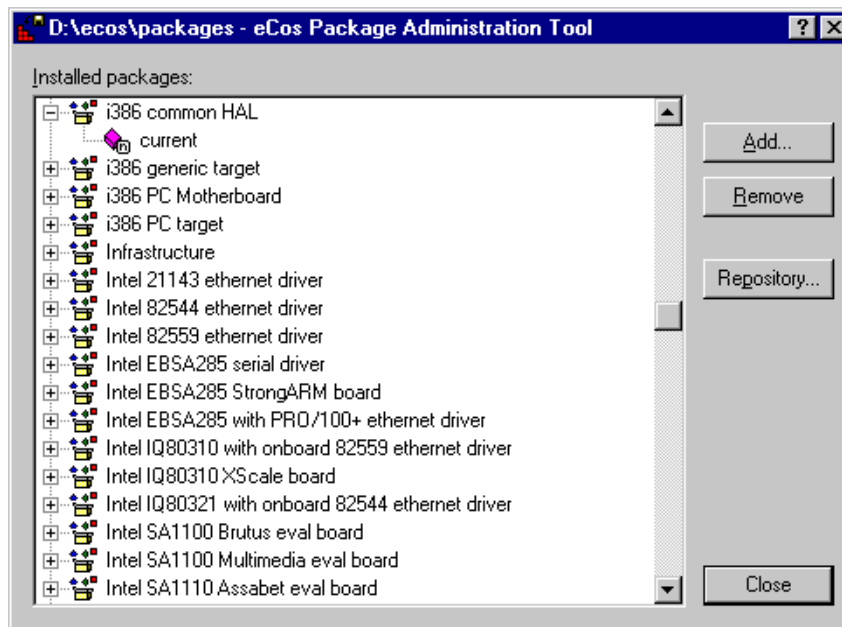
Each package in the eCos component repository is a self-contained released module. A package contains an individually versioned source code structure. This method supports individual package distributions or other third-party contributors.

The eCos Package Administration tool is a graphical interface, which runs on Windows, that provides the management of all packages in the component repository. A command-line interface can also be used to manage the packages in the repository. We focus on the use of the graphical Package Administration tool in this section.

The Package Administration Tool is installed as part of the eCos development kit installation covered in Chapter 10. The executable is located in the `D:\ecos\bin` directory. The Package Administration tool can also be started from the Configuration Tool through the *Tools* → *Administration* menu. The source code for this tool, along with the other eCos tools, is also installed with the eCos development kit under the `D:\ecos\host\tools\pkgadmin` subdirectory.

The Package Administration tool uses a Tcl script file, named `ecosadmin.tcl`, located in the `D:\ecos\packages` subdirectory. This file allows you to install or remove packages in

the eCos component repository. Packages are distributed in a single file with a .epk extension. Figure 11.16 shows a screen snapshot of the eCos Package Administration tool.



**Figure 11.16** The eCos Package Administration tool.

As we see in Figure 11.16, the left side of the Package Administration tool displays the currently installed packages. The package icons are the same as those found in the Configuration Tool, which are described in Table 11.1.

The plus and minus signs next to the Package icons allow you to expand or contract the packages. When the package is expanded, the versions of the different packages currently installed are displayed. We can see that in Figure 11.16, the i386 Common HAL package has the current version installed.

**NOTE** The Package Administration tool provides warnings when removing certain packages from the component repository. You need to take care in your selection of particular packages, because removing certain packages—for example, the eCos Common HAL package—might require you to reinstall the eCos development kit in order to get the package back into the repository.

Clicking the Add button brings up a dialog box that allows us to browse for a new eCos package file (.epk) to install into our local source code repository. The Remove button deletes the selected package from our local eCos repository. When we select the Remove button, a confirmation

dialog box is displayed to ensure that the proper package is removed. The Repository button allows us to select a different eCos repository to operate on.

### 11.3.2 The Command-Line Configuration Tool

Configuration of packages in the eCos repository for your specific application requirements can also be done with the eCos command-line tool. The command-line configuration tool is installed with the eCos development kit installation, which we did in Chapter 10.

The executable is located in the `D:\ecos\bin` subdirectory. The command-line configuration tool can be invoked from the bash shell prompt by running the file `ecosconfig.exe`. Invoking the command-line executable without any parameters displays the help message showing the different commands available. The source code for this tool is also installed with the eCos development kit under the `D:\ecos\host\tools\configtool` subdirectory.

Since the graphical Configuration Tool is able to run on both Windows and Linux, with version 2.0 and beyond, we do not go into the details of using the command-line configuration tool. It is much easier to use the Configuration Tool to set up, configure, and build the eCos library. The command-line configuration tool also does not offer the ability to interactively resolve conflicts that arise during the configuration process.

## 11.4 Building the eCos Tools

Yes, even the eCos development tools, including the graphical Configuration Tool, the Package Administration Tool, and the command-line configuration tool, are open source and, therefore, you can build these tools if you desire or need to take on that task. This gives you complete control of your embedded development environment—you are completely autonomous.

The source code for the eCos toolset is contained in the eCos repository under the `D:\ecos\host` directory. Also included in this directory are other testing tools and utilities. The latest source code for the tools is available in the online CVS repository. Additional information about building the eCos toolset can be found on the eCos home site, as well as information regarding building the graphical Configuration Tool, at:

<http://sources.redhat.com/ecos/ct2.html>

## 11.5 Additional Open-Source Tools

In this section, we briefly look at a couple of additional open-source tools that can be used to complete our embedded software development environment. The tools described in this section are often used by developers to aid in producing more robust software and, therefore, a more robust and reliable product.

This brief overview is to make you aware of other open-source tools that can be appended to the core eCos development tools to round out your software development system. Sources for getting additional information are included for you to investigate these tools on your own. All files needed to set up the tools are provided on the CD-ROM.



### 11.5.1 Source-Navigator

*Source-Navigator* is a code analysis and comprehension tool. Source-Navigator is an open-source project that can aid in understanding and reengineering complex software projects. By parsing through C, C++, Java, Tcl, [incr tcl], FORTRAN, COBOL, and assembly language source code, Source-Navigator builds a project database. The database includes useful information such as internal program structures, the location of function declarations, contents of class declarations, and details the relationships between program components. The home site for the Source-Navigator project can be found online at:

<http://sources.redhat.com/sourcenav>

Source-Navigator also includes a Software Development Kit (SDK). The SDK allows you to add new parsers to support additional languages, modify the graphical user interface, query the database for specific information, and use other applications in conjunction with Source-Navigator, such as a version control system. In addition, it is possible to configure Source-Navigator to build programs and launch the GNU debugger for a completely integrated development environment.

Source-Navigator version 5.0 is included on the CD-ROM under the `srcnav` directory. The file `sourcenavigator50.zip` contains the Windows executable you can use to set up Source-Navigator on your development system. Pre-built binary files can also be downloaded for Solaris and HPUX.

Simply unzip the `sourcenavigator50.zip` file, retaining the directory structure, onto your `D:\` drive. Source-Navigator requires approximately 17.4 Mbytes of disk space. All necessary files are placed under the directory `D:\SourceNavigator`. To start Source-Navigator, run the executable file `snavigator.exe` under the `bin` subdirectory.

Also included on the CD-ROM is the source code for the version 5.0 release of Source-Navigator. The source code is contained in the file `SN50-010322-source.tar.gz` also under the `srcnav` directory.

The *Source-Navigator User's Guide* includes details about using the project editor, using the symbol browser, customizing Source-Navigator for your specific needs, and other features, as well as a tutorial. The *User's Guide* can be found online at:

[http://sources.redhat.com/sourcenav/online-docs/userguide/index\\_ug.html](http://sources.redhat.com/sourcenav/online-docs/userguide/index_ug.html)

The *Source-Navigator Programmer's Reference Guide* contains information about adding parsers, the database API, and integrating a version control system into Source-Navigator. The *Programmer's Reference Guide* can be found online at:

[http://sources.redhat.com/sourcenav/online-docs/progref/index\\_pr.html](http://sources.redhat.com/sourcenav/online-docs/progref/index_pr.html)

Two discussion mailing lists are available for Source-Navigator. The first is a general discussion list for issues and submission of patches, which is located online at:

<http://sources.redhat.com/ml/sourcenav>

Posts for this list should be sent to:

`sourcnav@sourceware.cygnum.com`

The other mailing list is found online at

`http://sources.redhat.com/ml/sourcnav-announce`

This is an announcement list for posts about new releases or other important information. Posts for the announcement list should be sent to:

`sourcnav-announce@sourceware.cygnum.com`

### 11.5.2 Splint

No, lint is not referring to the stuff you find in the deepest recesses of your pockets. A lint program is used to statically verify a program, or part of a program, against standard libraries, checks the code for portability, and checks the code for common errors such as ignored return values, unused declarations, and type inconsistencies.

Although a compiler provides some checking, lint checks these areas of a program much more carefully and provides output messages where possible problems may occur. Lint can be used to develop more robust software.

Splint is an open source development project; therefore, it can be used free of charge. Splint stands for Secure Programming Lint or Specifications Lint. Additional information about Splint can be found online. The home site for Splint is located at:

`www.splint.org`

Included on the CD-ROM under the `splint` directory is a zip file, `splint-3016win32.zip`, which contains the Splint executable version 3.0.1.6, as well as documentation on how to set up and use the program. A Linux version of Splint is also included on the CD-ROM under the `splint/linux` directory in the file `splint-3.0.1.6.Linux.tgz`.

Also included on the CD-ROM under the `splint` directory is the source code for this Splint version. This is located in the file `splint-3.0.1.6.src.tgz`. Instructions for building Splint can be found online at:

`www.splint.org/source.html`

To install Splint, unzip the file `splint-3016win32.zip`, retaining the directory structure, onto your `D:\` drive. This will place all Splint files under the directory `D:\splint`. The executable is located in the `bin` subdirectory. Two environment variables need to be set as follows:

- `LARCH_PATH="D:\splint\lib"`
- `LCLIMPORTDIR=D:\splint\imports`

Finally, set up the command path to include the directory containing the `splint.exe` executable. Additional installation information can be found in the file `README` file or on the Splint home site.

## 11.6 Summary

In this chapter, we focused on the different eCos development tools. We started with an overview of the CDL used in the eCos framework. Moving onto the Configuration Tool, we discovered the relationship between the CDL files and the graphical representation of packages in the Configuration Tool. We also examined the eCos database file and how it is used by the Configuration Tool to represent the information in the repository.

A typical eCos configuration involves selecting one target and one template. Packages can then be added and/or removed based on the requirements of the system. Then, fine-grained configuration is performed on the configuration options.

Then we moved on to the other tools available in the eCos development kit, as well as other open-source tools that allow us to configure a complete embedded development environment free of charge since the tools are all open source. Now that we have an understanding of the eCos Configuration Tool, we are able to move to the next step, which is using the eCos tools to develop our own application.

# An Example Application Using eCos

In this chapter, we get down to using eCos and the development tools. The examples here detail one method for setting up a debug environment using RedBoot, building an eCos library, and then building an application that incorporates eCos. There are several different approaches to use to achieve a similar development and debug environment using different configuration options. These examples provide a straightforward method to achieving our goal: getting eCos up and running.

We begin by building the RedBoot ROM monitor, which allows us to debug our application using GDB on our host development system. Next, we build an eCos library image for us to use with our application. Then, we build a simple application, which includes the eCos library image, to understand the process. Each of the examples in this chapter builds on the previous steps. After becoming familiar with the process, you will be able to tailor the build and debug procedure to meet your own needs and development style.

## 12.1 The eCos Build Process

The main goal of the eCos build process is the generation of an eCos library. This library is called `libtarget.a`. Other target files are also generated, such as a linker script file and, in some cases, additional libraries may be generated.

The RedBoot build process is slightly different from that for eCos because a binary file is generated at the end of a RedBoot build, which is then installed on the target hardware.

The eCos library build process uses makefiles and the GNU make utility to assist in generating the eCos library. The eCos build process involves three separate trees: source, build, and install. The source tree is the source code repository, which is located under the `packages` directory.

The build tree is generated by the configuration tools and contains intermediate files, such as makefiles and object files. The structure of the build tree might differ between system builds. Typically, each package in the configuration has its own directory in the build tree, which is used to store that package's makefiles and object files.

The install tree is the location of the eCos main library file and the exported header files, which are used when the application is built. The library files are located under the `lib` subdirectory, and the header files are contained under the `include` subdirectory. By default, the build and install trees are contained in the same working directory.

### 12.1.1 A Closer Look

Let's take a closer look at the eCos build process using our example eCos configuration that we cover later in this chapter. In this example, we are building an eCos library for use on the i386 PC hardware platform.

The build process starts with a configuration. We are able to construct a configuration according to our requirements using the Configuration Tool, which allows us to incorporate certain packages and set configuration option values. After we have included the packages we need and set the configuration options accordingly, we save our configuration file in the file `ecos.ecc`. This generates the build and install tree in our working directory. A detailed look at the build and install tree is shown in Figure 12.6 later in this chapter.

By saving our configuration, the Configuration Tool generates the files needed for our build process. This includes makefiles, located in the build tree, and header files, located in the install tree.

For now let's focus on the i386 HAL package and how the configuration option settings are used to generate files for the build process. Figure 12.1 shows a portion of the Configuration Tool configuration window, along with the generated build files that match the configuration option settings.

As we see in the upper-left corner of Figure 12.1, the i386 Architecture package (CYGPKG\_HAL\_I386) is displayed from the Configuration Tool. The Enable I386 FPU Support (CYGHWR\_HAL\_I386\_FPU) configuration option is enabled, while the other two configuration options are disabled. The following descriptions correspond to the numbers in Figure 12.1. The ellipsis ( . . . ) in the code excerpts denote places that the code was cut out to allow us to focus on specific areas of the source files.

1. First, we see the i386 Architecture package represented in the CDL script file `hal_i386.cdl`, which is part of the source tree. Since this package is loaded into the configuration, the Configuration Tool uses the i386 Architecture package CDL script file to generate the necessary build files.

The `cdl_package` command for the `CYGPKG_HAL_I386`, shown on line 1, contains the files that need to be compiled on line 3, designated by the `compile` property. The `compile` property is used by the Configuration Tool at build time to determine

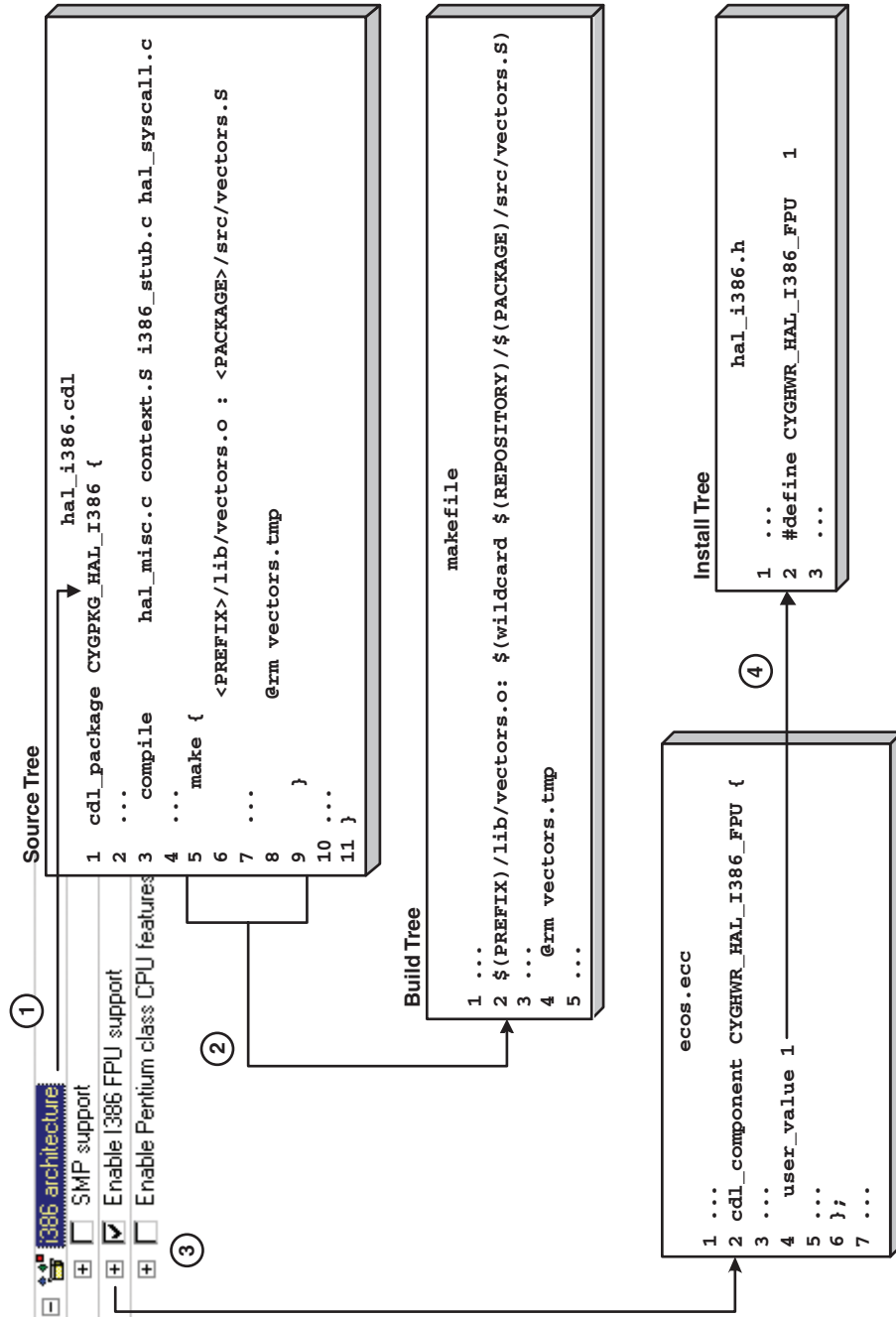


Figure 12.1 Configuration Tool file generation diagram.

which files need to be compiled in order to include the features of the i386 Architecture package. On lines 5 through 9 are the `make` properties, which are used by the Configuration Tool to generate the appropriate makefile used to compile the i386 Architecture package.

2. Next, we see a portion of the `makefile` generated by the Configuration Tool, which occurs when we save our configuration. This `makefile` is part of the build tree and is located in our working directory under `ecos_build\ hal\i386\arch\current`. Details about the build and install tree are covered later in this chapter. In Figure 12.1, we see the translation from the CDL script file `make` property to the generation of the `makefile` on lines 2 through 4.
3. Now we see that the Enable I386 FPU Support (`CYGHWR_HAL_I386_FPU`) configuration option is enabled in the Configuration Tool. When the configuration is saved in the `ecos.ecc` configuration file, the CDL script is generated as we see on lines 2 through 6.
4. Finally, the Configuration Tool generates a header file `hal_i386.h` that includes the `CYGHWR_HAL_I386_FPU` configuration option. This header file is located in the install tree portion of our working directory under the `ecos_install\include\pkgconf` directory. Since this option is enabled, the Configuration Tool defines the configuration option `CYGHWR_HAL_I386_FPU` as 1, which we see on line 2 of this file excerpt.

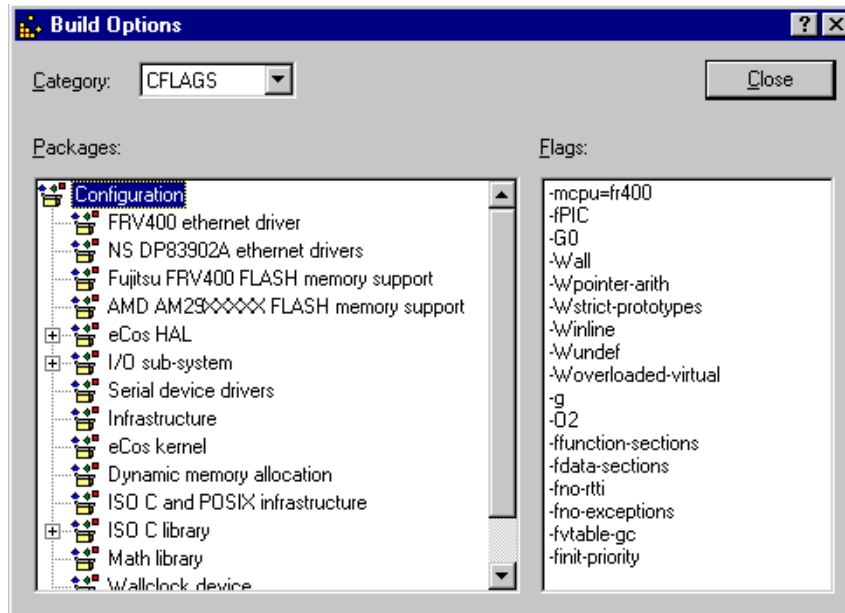
The preceding example shows the simple generation by the Configuration Tool of two files for the configuration setup. After all files are generated and the appropriate header files are included in the install tree, the build process can continue.

Next, the relevant source files, based on the packages included in the configuration, are compiled. The Configuration Tool determines which files need to be compiled from the CDL script file, as shown in Figure 12.1 in the `hal_i386.cdl` file. The `compile` property alerts the Configuration Tool of the necessary files for including the specified package's functionality into the build.

Global compiler flags (`CYGBLD_GLOBAL_FLAGS`) are used during the compilation stage of the build process. Some packages have their own specific build options as well. In addition, in some cases, packages allow certain global compiler flags to be suppressed. The compiler flags can be set just as the other configuration options.

During the build process, the object files are output into the build tree. The build tree is structured according to packages, similar to the source tree. Then, the Configuration Tool links the files together. Global linker flags (`CYGBLD_GLOBAL_LDFLAGS`) can be set, just as the compiler flags.

The compiler flags and linker flags can also be displayed in a dialog box in the Configuration Tool. An example of this dialog box is shown in Figure 12.2.



**Figure 12.2** Configuration Tool build options dialog box.

The dialog box in Figure 12.2 is displayed by selecting *Build* → *Options*. The compiler (CFLAGS) or linker flags (LDFLAGS) are selected using the drop-down list in the upper-left corner of the dialog box. The left pane displays the packages in the configuration. The right pane shows the flags for the specific package selection. Different flags can be displayed by selecting different packages in the left pane. In Figure 12.2, the global compiler flags are displayed since the entire configuration is selected in the Packages pane.

As the final step in the build process, for an eCos image, the Configuration Tool invokes the archive utility to create a library file. This library file is output into the install tree under the `ecos_install\lib` directory. In the case of building a RedBoot image, the final product is a binary file, which is output in the install tree under the `ecos_install\bin` directory.

Additional details about each step in the eCos build process, as well as information on how to customize the build process, are provided in the *eCos Component Writer's Guide*. This document can be found online at:

<http://sources.redhat.com/ecos/docs.html>

## 12.2 Examples Overview

Several different approaches can be used to configure a target system to load and debug eCos applications. The method described here offers a direct approach to using the different tools available in the eCos system. Each platform has its own techniques for loading and debugging

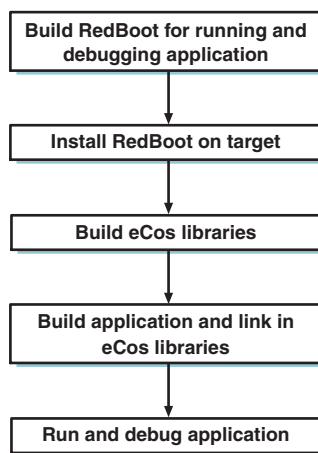


code. Using the examples in this chapter, you will have a solid understanding of the development process using the eCos system.

The examples in this chapter assume that one of the supported hardware platforms is used for the target system. Moving to your own hardware platform and getting eCos running is the end goal, which we discuss in Chapter 13, *Porting eCos*.

An overview of the steps involved in building and running the examples in this chapter is shown in Figure 12.3.

**Figure 12.3** Overview of the stages for examples covered in this chapter.



As we see in Figure 12.3, the first stage covered in the examples in this chapter is to build the RedBoot ROM monitor to provide application load and debug support. Next, we install the RedBoot image onto the target hardware. These first two stages are covered in the *RedBoot* section.

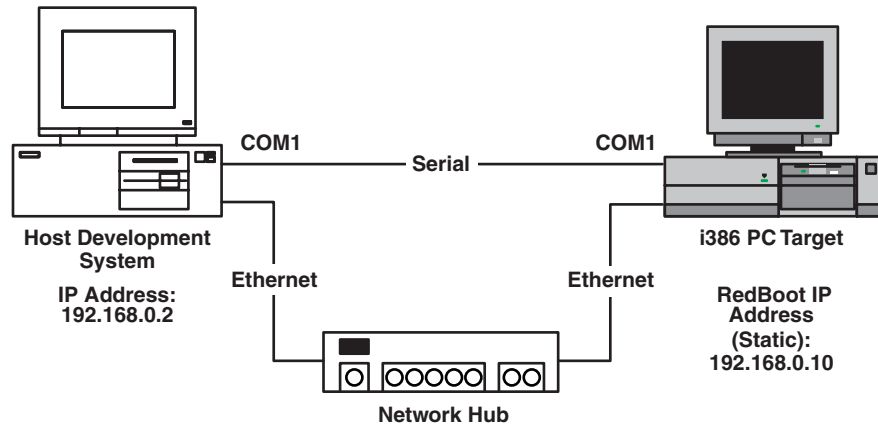
The next step is to configure eCos and build the libraries according to our configuration settings, which is covered in the *eCos* section. These libraries are linked in by the application when the application is built.

Now we can build our example application using the eCos libraries from the previous stage. This application is a simple example showing some of the functionality provided by eCos. After we have an application image, we can run and debug the image using RedBoot and GDB. These two stages are covered in the *Application* section.

### 12.2.1 Development Hardware Setup

The development configuration used for the examples in this chapter, which shows the PC platforms and other hardware modules, is detailed in Figure 12.4.

Figure 12.4 gives us an overview of the major components used in the development environment for the examples in this chapter. The Host Development System contains the complete eCos development environment, which we set up in Chapter 10, *The Host Development Platform*. This system is also the host when running GDB during debug sessions.



**Figure 12.4** Development environment configuration for eCos examples.

The i386 PC Target is a Pentium-based PC used to run RedBoot and our applications. The target system also needs to have a floppy drive. As we see in Figure 12.4, the host connects to the target via serial and Ethernet. We can use either connection for console and debug communication between the host and target, which was covered in Chapter 9, *The RedBoot ROM Monitor*.

Finally, a network hub is used to connect the Ethernet ports together between the host and target system. The diagram gives the IP addresses used for the examples in this chapter. It might be necessary in your own development environment to use different IP addresses based on your network configuration. The points where the network is configured are detailed in the examples.

When using the serial port for debug and communications a null modem cable is used to connect the COM port of the host to the serial port on the target.

The Ethernet cards used in both PCs use the Intel 82559 Ethernet controller. It is important to use an Ethernet card that contains a controller supported by eCos; otherwise, the Ethernet port might not work properly for communications.

In this development environment, the IP addresses are statically configured. This eliminates the need for a DHCP or BOOTP server, which simplifies the development environment. The Host Development System uses IP address 192.168.0.2. The RedBoot IP address is 192.168.0.10. Information regarding RedBoot and static IP addresses is covered in Chapter 9. By using these IP addresses, a private network is created. Both connections are on the same network using a subnet mask of 255.255.255.0.

---

**NOTE** It is not a good idea to use static IP addresses in a released product. The network that a device goes into can vary widely. In this case, it is better to use a dynamic IP address configuration scheme such as DHCP or BOOTP in order for the device to obtain its IP address. However, we use static IP addresses in these example scenarios to simplify things.

### 12.2.2 eCos Tools

The Configuration Tool, instead of the command-line interface, is used to configure and build the images needed for the examples in this chapter. Other tools that are used are also detailed in the examples in this chapter.

---

**NOTE** The Configuration Tool is evolving just as the eCos source code itself. Since this is the case, there are times when the Configuration Tool does not behave exactly as expected. In these cases, a note is given describing the circumstance that might occur and workarounds are detailed to ensure that you can proceed with the examples successfully.

The examples in this chapter use a working directory to store the build and configuration files. The working directory is `D:\workdir` used for each example.

All software needed to recreate the development environment used for building and running these examples are contained on the CD-ROM under the `examples` directory. These files offer a good starting point to bring up your own development environment, which you can then augment to meet your development and debugging requirements.

## 12.3 RedBoot

In this section, we go through the process of building, loading, and booting RedBoot. There are pre-built binary images of the RedBoot ROM monitor. These are located in the online eCos repository under the subdirectory `ecos\images`. The RedBoot images are grouped according to hardware architecture. It might suffice to use these images in the beginning; however, understanding how to build your own RedBoot image is important if you need to make bug fixes or enhance the standard RedBoot program.

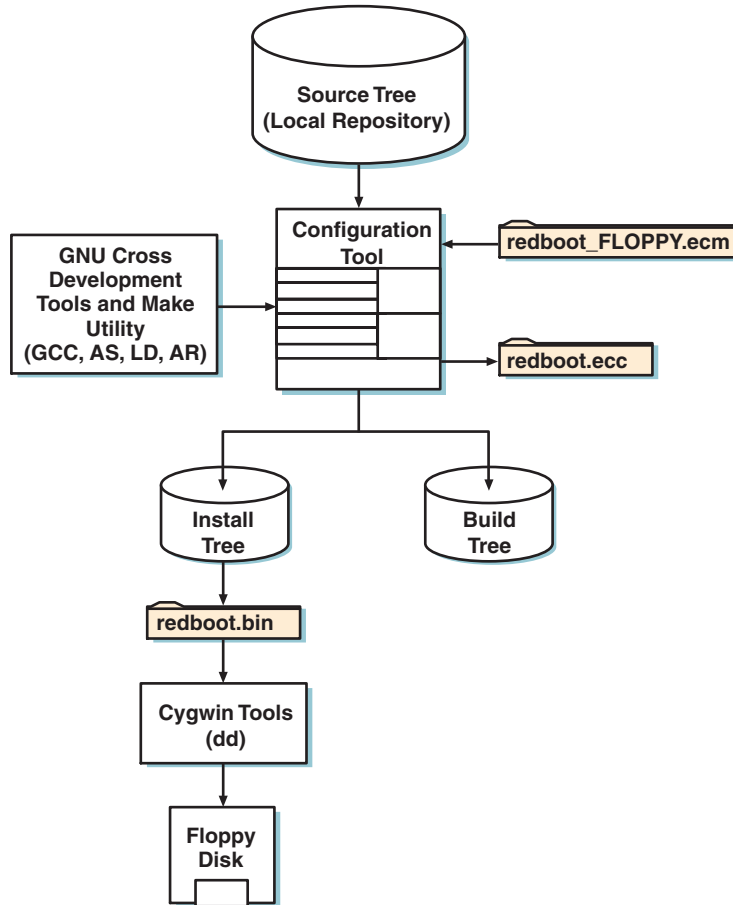
The RedBoot image built in this example uses the FLOPPY Startup Type configuration option. This means that the RedBoot image boots and runs from the floppy disk drive of the target PC.

### 12.3.1 Building RedBoot

As mentioned before, we are going to use the Configuration Tool to configure and build RedBoot. Since we are using RedBoot to debug our example application, we need to configure the IP addresses properly. We discussed IP address configuration with respect to RedBoot in Chapter 9.

Before we begin, let's get an overview of the build procedure. Figure 12.5 shows us the flow of the build and install procedure for our RedBoot example.

As we see in Figure 12.5, the beginning of the RedBoot build procedure starts with the source tree; in our case, this is located under `D:\ecos\packages`. The appropriate files are included into our configuration, which we store in the file `redboot.ecc`.



**Figure 12.5** RedBoot build and install procedure flow diagram.

Next, the Configuration Tool uses the GNU cross-development tools, such as `i386-elf-gcc`, to build our RedBoot image. The output from the Configuration Tool is stored in the build and install trees. Our final RedBoot image, `redboot.bin`, is the product of our build procedure. Finally, this image is installed onto a floppy drive using the Cygwin tools.

In our examples, we are going to use a static IP address for RedBoot. The static IP address used depends on the specific network configuration where the target hardware resides. The hardware environment for the examples, including the IP address configuration, is shown in Figure 12.4.

The CD-ROM contains an eCos configuration file (`.ecc`) that was saved after going through the steps to build the RedBoot image. The RedBoot configuration file is `redboot.ecc`, which is located under the `examples\redboot` subdirectory. This file is provided in case you need to recreate the exact configuration file used for this example. Also included in this directory

on the CD-ROM is the entire install tree under the directory `examples\redboot\redboot_install` created from the example RedBoot build procedure. The binary files created (`redboot.bin` and `redboot.elf`) are in the `bin` directory under the install tree. The MLT directory is also included under `examples\redboot\redboot_mlt`.

### STEP 1

The first step is to load the packages we need to build the RedBoot image using the Configuration Tool. For this, we use a template. The template dialog box is launched by selecting *Build* → *Templates*.

In the templates dialog box we select `i386 PC Target` from the hardware drop-down list. From the packages drop-down list we select the `redboot` package. Then, click the OK button.

The Resolve Conflicts dialog box might pop up because we are changing to a new template with different configuration option settings. Using the Configuration Tool, we can resolve these conflicts automatically. We want to click the Continue button to proceed with loading the proper packages and configuration options for the `i386 PC Target`.

### STEP 2

Next, we want to set up the configuration options for our RedBoot build. We can do this by manually selecting the various configuration option settings, or by importing a pre-configured eCos minimal configuration file (`.ecm`). The `.ecm` files are included in the source code repository for each HAL platform supported, which give us a baseline for configuration settings needed to build a valid image.

To import the eCos minimal configuration file, we select *File* → *Import*. Now we browse to the location of the `i386 PC` platform `redboot_FLOPPY.ecm` file. This file is located under the `D:\ecos\packages\hal\i386\pc\current\misc` subdirectory. After selecting the `redboot_FLOPPY.ecm` file, we click the Open button.

The resolve conflicts dialog box might pop up. Click the Continue button to proceed with importing the eCos minimal configuration file.

### STEP 3

Now we need to verify the configuration option settings. First, we want to ensure that the RedBoot IP address is configured properly.

The RedBoot IP address configuration options are located within the *RedBoot ROM Monitor* (`CYGPKG_REDBOOT`) package under the *RedBoot Networking* (`CYGPKG_REDBOOT_NETWORKING`) package. The configuration option we want to enter is the *Default IP Address* (`CYGDAT_REDBOOT_DEFAULT_IP_ADDR`). The static IP address for the RedBoot image is `192,168,0,10`. We also want to enable the *Do Not Try To Use BOOTP* (`CYGSEM_REDBOOT_DEFAULT_NO_BOOTP`) configuration option.

---

**NOTE** The *Default IP Address* configuration option must be entered with commas separating the IP address numbers, not decimals.

We also want to verify the configuration of the communication channels on the target system. The communication channels are nested under the *eCos HAL* package, under *i386 Architecture* (CYGPKG\_HAL\_I386) package, under the *i386 PC Target* (CYGPKG\_HAL\_I386\_PC) platform package. The configuration suboptions under the *i386 PC Target* package control the settings for the communication channels. The *Number of Communication Channels on the Board* (CYGNUM\_HAL\_VIRTUAL\_VECTOR\_COMM\_CHANNELS) configuration option defines the PC serial ports that are used to communicate with the host. The default is set to 3 for this option. The mapping of these communication channels to PC ports is detailed in Table 12.1.

**Table 12.1** Communication Channel to PC Port Mapping

| Communication Channel | PC Port                |
|-----------------------|------------------------|
| Channel 0             | COM 1                  |
| Channel 1             | COM 2                  |
| Channel 2             | PC Screen and Keyboard |

The communication channels for debug (*Debug Serial Port* [CYGNUM\_HAL\_VIRTUAL\_VECTOR\_DEBUG\_CHANNEL]) and console (*Default Console Channel* [CYGNUM\_HAL\_VIRTUAL\_VECTOR\_CONSOLE\_CHANNEL\_DEFAULT]) are configured to use channel 0, or COM 1. However, the communication channel RedBoot receives input on first is used as the communication channel.

Since we have the configuration option *Output to PC Screen* (CYGSEM\_HAL\_I386\_PC\_DIAG\_SCREEN) enabled, you can use the PC monitor and keyboard for communication with RedBoot.

#### STEP 4

Next, we save our configuration file. For this, we select *File -> Save As*. We need to pick a name for the eCos configuration file (.ecc) that we are saving. We use the filename `redboot.ecc`.

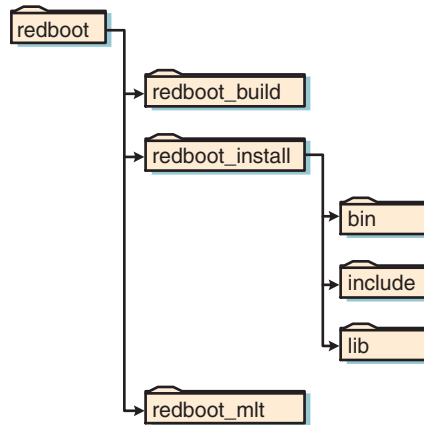
Now, we browse to our working directory location where the build and install trees are generated. We use `D:\workdir\redboot`. If this directory does not exist, you can create it now. Then, click the Save button.

The Configuration Tool takes the .ecc filename selected, `redboot` in our case, and creates the directories needed to build the image we have configured. The Configuration Tool appends `_build` and `_install` to the filename when generating the build and install tree directories.

Figure 12.6 shows the working directory structure created by the Configuration Tool for the RedBoot image we are building.

The `redboot_build` subdirectory contains the build tree and is used by the Configuration Tool to store the different files used and created in the build process, such as makefiles and

**Figure 12.6** Working directory structure for RedBoot example.



object files. The `redboot_install` subdirectory contains the install tree and includes the final output binary images as well as the header files used for the build process. The `redboot_mlt` subdirectory contains the memory layout files used by the Configuration Tool Memory Layout Tool.

### STEP 5

Now we have the packages and configuration options settings loaded that we need to build the RedBoot image for the i386 PC platform. We can now proceed with building the image. We want to verify that the status bar, in the lower right-hand corner, on the Configuration Tool shows No Conflicts. If conflicts were present in our configuration, we could view them by opening the Conflicts window by selecting *View -> Conflicts*, or using the hot key *Alt+5*.

To start the build, select *Build -> Library*, or click the Build Library icon on the toolbar. During the build process, the Configuration Tool shows messages on the status bar. In this case, Building is displayed. The output window also displays messages as the build progresses.

When the RedBoot build is complete, the output window displays `build finished`. If build errors were to occur, the build process would halt and specific error messages would be displayed in the output window. After the errors are corrected, selecting *Build -> Library* again proceeds with the build process.

The file we need is located under `redboot_install\bin` subdirectory. We use the `redboot.bin` file.

### 12.3.2 Installing RedBoot

The method for installing the RedBoot image on the target hardware varies from platform to platform. The next steps detail the process for booting an image from a floppy disk drive. Using other Startup Types or hardware platforms might require a different procedure. A floppy disk is required to install and run RedBoot using the floppy drive.

Additional information about installing RedBoot on the various supported hardware platforms can be found in the RedBoot documentation online at:

<http://sources.redhat.com/ecos/docs.html>

### STEP 6

Launch the *Cygwin Bash Shell* by double-clicking the Cygwin desktop icon or selecting *Cygwin Bash Shell* under *Start -> Programs -> Cygnus Solutions -> Cygnus Solutions*. Using the bash shell, we enter commands to mount the floppy drive and install the binary image onto a floppy disk.

First, we need to make sure that the floppy drive is mounted properly on the development system. For this, we enter the command:

```
$ mount -f -b ../a: /dev/fd0
```

### STEP 7

Next, we want to make sure we are in the proper subdirectory to install the RedBoot binary image. We want to change to the proper working directory in the bash shell using the command:

```
$ cd d:/workdir/redboot
```

### STEP 8

Now, we place a floppy disk into our host development system's floppy disk drive. We are using the `dd` utility, which was installed during the Cygwin installation procedure, to transfer the RedBoot image to the floppy disk.

The `dd` utility writes raw data from the standard input to standard output. In our case, we write the RedBoot binary file from the hard disk to our floppy disk.

---

**NOTE** The process of transferring the RedBoot image to the floppy disk erases any file system and data contained on the disk.

To install the RedBoot binary image onto the floppy drive we use the command:

```
$ dd conv=sync if=redboot_install/bin/redboot.bin of=/dev/fd0
```

After the process is complete, a message is displayed showing the records converted, similar to:

```
178+1 records in
179+0 records out
```

The operation is complete when the bash shell returns the `$` prompt.

### 12.3.3 Booting RedBoot

We are using the development environment configuration as shown in Figure 12.3. Using this configuration, we are able to debug applications over the serial port or Ethernet port, depending on the functionality in the application.



For this example, we are using the Windows resident terminal program called HyperTerminal, although any terminal program can be used. Remember, RedBoot uses the first communication channel it receives input from as its port to communicate with the host.

### STEP 9

Place the floppy disk containing the RedBoot image into the drive on the target PC. Power up the PC. Since our configuration allows output to the PC monitor (as described in step 3), RedBoot outputs the initialization message shown in Code Listing 12.1 to the screen.

```
1 Ethernet eth0: MAC address 00:d0:bd:43:9d:d2
2 IP: 192.168.0.10, Default server: 0.0.0.0, DNS server IP: 0.0.0.0
3
4 RedBoot(tm) bootstrap and debug environment [FLOPPY]
5 Non-certified release, version UNKNOWN - built 12:22:06, Apr 20
 2002
6
7 Platform: PC (I386)
8 Copyright (C) 2000, 2001, 2002, Red Hat, Inc.
9
10 RAM: 0x00000000-0x000a0000, 0x0008ac30-0x000a0000 available
11 RedBoot>
```

**Code Listing 12.1** RedBoot initialization message.

Code Listing 12.1 shows us the initialization message after booting RedBoot. The RedBoot network configuration information is displayed on lines 1 and 2, including the static IP address we configured in step 3, 192.168.0.10. Lines 4 and 5 give us information about the RedBoot image, including the floppy startup and the build time and date. The platform information is output on line 7. The RAM memory utilization and availability is output on line 10. Finally, the RedBoot prompt is output on line 11 showing that RedBoot is ready for input.

### STEP 10

Next, we connect to the target PC using HyperTerminal. The communication parameters we set in HyperTerminal are a baud rate of 38400, 8 data bits, no parity, 1 stop bit, and no flow control. After clicking connect, we can press Enter to get a RedBoot prompt displayed.

We can verify communication over the serial port by entering the `help` command, to which RedBoot responds with the commands available.

### STEP 11

Finally, we verify our network connections using the ping utility. We verify the connection between our host and the target. We accomplish this by entering the following command at the RedBoot prompt:

```
RedBoot> ping -v -n 3 -h 192.168.0.2
```

The preceding ping command is set in verbose mode, using the `-v` option; the number of ping packets to send is set to 3 by the `-n` option; and the address to send the ping packets is set to `192.168.0.2` by the `-h` option. The IP address you use depends on your network configuration. This results in the output shown in Code Listing 12.2.

```
1 Network PING - from 192.168.0.10 to 192.168.0.2
2 seq: 1, time: 1 (ticks)
3 seq: 2, time: 1 (ticks)
4 seq: 3, time: 1 (ticks)
5 PING - received 3 of 3 expected
6 RedBoot>
```

**Code Listing 12.2** Response output from ping command to host PC.

On line 1 of Code Listing 12.2, RedBoot displays the message that the ping command is executing and the two IP addresses being used. Lines 2 through 4 output the response sequence numbers and times, in ticks, from the ping packets. Finally, line 5 gives a summary of the successful reception and response of the three ping packets sent from the target to the host.

A telnet program can also be used to connect to the target PC via the Ethernet port. This allows you to use telnet to send commands to and receive output from RedBoot, in the same way as using the serial port. One thing to remember is that the telnet connection must be on port 9000, as set by the *TCP Port To Listen For Incoming Connections* (`CYGNUM_REDBOOT_NETWORKING_TCP_PORT`) configuration option under the RedBoot package.

---

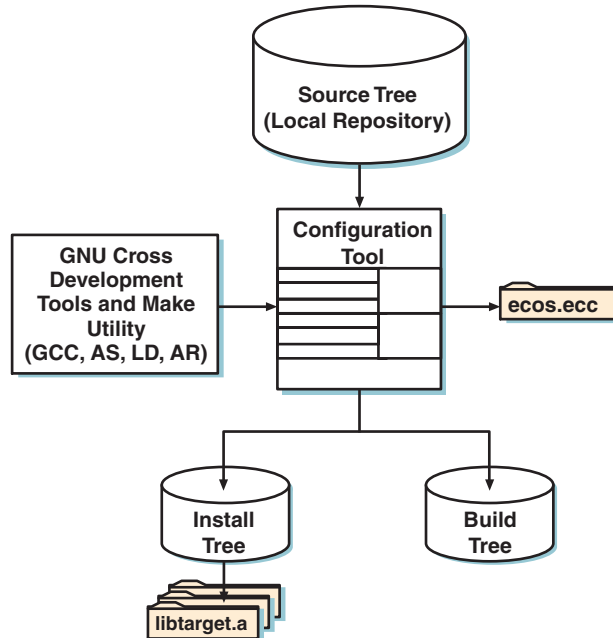
**NOTE** Although this is not true for the PC platform booting from the floppy drive, other hardware platforms allow RedBoot images to be installed into flash ROM memory. In these cases, constantly reprogramming the flash device can be tedious. Therefore, RedBoot can be used to update an older image running from flash memory. Information about updating existing RedBoot images residing in flash memory on various target platforms can be found in the RedBoot documentation online at <http://sources.redhat.com/ecos/docs.html>.

## 12.4 eCos

Continuing with our examples, we now need to configure and build an eCos image that will be linked with our application. The eCos image we build in this section uses the RAM Startup Type configuration option. This allows us to use RedBoot to load our example application into RAM. We are then able to run and debug our application.

### 12.4.1 Building eCos

The main goal of this stage is to generate the eCos library file according to our configuration settings. The flow diagram for the eCos build procedure used in this example is shown in Figure 12.7.



**Figure 12.7** eCos build procedure flow diagram.

Similar to the RedBoot build procedure, we start with the source tree as shown in Figure 12.7. After setting up our configuration according to our specification using the Configuration Tool, we save the file `ecos.ecc`.

Next, the Configuration Tool generates the appropriate files for our build. The GNU cross-development tools are used to compile the source code files and produce our final output file, the eCos library `libtarget.a`. Other necessary files, which include additional libraries and a linker script file, are also produced by the Configuration Tool.

The CD-ROM includes the eCos configuration file (`.ecc`) containing the packages and configuration option settings used in this example. The filename is `ecos.ecc` and is located in the `examples\ecos` subdirectory. As with the RedBoot image, you can recreate the eCos library files by copying this file to your working directory and then loading it into the Configuration Tool.

Also included in this directory on the CD-ROM is the install tree from this example eCos build procedure. These files are located under the `examples\ecos\ecos_install` directory. The MLT directory is also included under `examples\ecos\ecos_mlt`.

---

**NOTE** It is a good idea to start with a new configuration by selecting *File* → *New* from the menu before proceeding with the eCos build procedure. This resets the Configuration Tool to its default platform and default configuration option settings, and clears out any previous configurations.

### STEP 1

We need to load the packages to build the eCos image. We use a template to get a baseline of packages loaded to meet our requirements.

The template dialog box is launched by selecting *Build -> Templates*. In the Templates dialog box, we select *i386 PC Target* from the hardware drop-down list. From the packages drop-down list we select the `default` package. This selects the standard packages for incorporation into the eCos image. Then, click the OK button.

The Resolve Conflicts dialog box might pop up because we are changing to a new template with different configuration option settings. Using the Configuration Tool, we can resolve these conflicts automatically. We want to click the Continue button to proceed with loading the proper packages and configuration options for the *i386 PC Target*.

### STEP 2

Now that we have a baseline configuration for our eCos image, we want to verify the configuration option settings. You might need to tailor the configuration option settings according to the specific target hardware you are using. Some of the configuration options might be set to the proper values for our build by default. In this case, no modifications are necessary.

First, we want to verify the *Startup Type* (`CYG_HAL_STARTUP`) configuration option setting. This is nested under the *eCos HAL* package, under *i386 Architecture* (`CYGPKG_HAL_I386`) package, under the *i386 PC Target* (`CYGPKG_HAL_I386_PC`) platform package. We want to set the Startup Type to RAM, so that we can load our application into RAM for running and debugging.

Next, since we are using the RedBoot ROM monitor, we want to verify the *ROM Monitor Support* components. Specifically, we want to enable the *Work With a ROM Monitor* (`CYGSEM_HAL_USE_ROM_MONITOR`) configuration option. Since we are going to use GDB on our host system for debugging, we want to set this option to `GDB_stubs`.

It is a good idea to enable asserts during the debug phase of a project. This enables various levels of checking in the software, such as argument checking, to ensure the software is behaving properly. As the software gets closer to a production release asserts should be disabled to ensure there are no timing problems hiding in the system. Asserts are enabled by selecting the *Asserts & Tracing* (`CYGPKG_INFRA_DEBUG`) component and then enabling the *Use Asserts* (`CYGDBG_USE_ASSERTS`) configuration option, which is enabled by default. This configuration option is located under the *Infrastructure* package.

Other configuration options might need to be validated or modified for your specific target hardware. This step is intended to make us aware of some of the key configuration option settings necessary to build an eCos library that works for our examples.

### STEP 3

Next, we save our configuration file by selecting *File -> Save As*. We use the filename `ecos.ecc`.

Now, we want to browse to our working directory where we want the eCos files stored. We use `D:\workdir\ecos`. If the directory does not exist, you can create it now. Then, click the Save button.

The Configuration Tool saves the file `ecos.ecc` under the new eCos working directory and creates the build and install tree directories named `ecos_build` and `ecos_install`, respectively. The Configuration Tool MLT file is contained in the `ecos_mlt` directory. The directory structure for this build process is similar to that shown in Figure 12.4.

#### STEP 4

We are now ready to build the eCos library. Prior to building, we ensure that there are no conflicts in our configuration, which is shown in the bottom-right corner on the status bar. If conflicts were present in our configuration, we could view them by opening the Conflicts window by selecting *View -> Conflicts*, or using the hot key *Alt+5*.

To start the build, select *Build -> Library*, or click the Build Library icon on the toolbar. During the build process, the Configuration Tool shows the Building message on the status bar. The output window also displays messages as the build progresses. After the build completes, the output window displays `build finished`.

If build errors were to occur, the build process would halt and specific error messages would be displayed in the output window. After the errors are corrected, selecting *Build -> Library* again continues the build process.

The files that we need for the next application example are located in the `ecos_install` subdirectory. The header files used when we build our application are contained under the `ecos_install\include` directory. The eCos library files and linker script are located under the `ecos_install\lib` directory.

We are now ready to proceed with building our application.

---

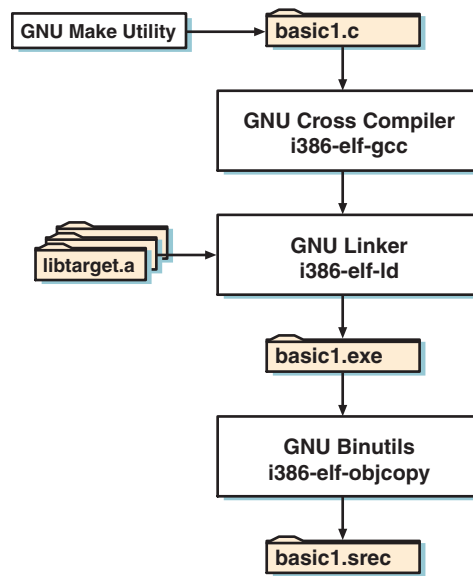
**NOTE** If modifications need to be made to a configuration after it is built, the Configuration Tool can be used to remove the build files generated. This is done by selecting *Build -> Clean* from the menu. However, there are times when this does not correctly remove all files and can cause problems in the resulting output image. To be safe, it is best to delete the build and install trees to ensure that all files are rebuilt for an image. In our example, these directories are `ecos_install` and `ecos_build`, which we would want to delete. Saving the configuration causes the Configuration Tool to regenerate these directories for the next build.

## 12.5 Application

We are now at the point where we can build our application and incorporate our eCos library built in the previous stage. After we have our application image, we can use RedBoot to load the application into RAM and run it. In this section, we also look at using GDB to debug our application.

The main goal of the application build is to generate a binary file, which includes our eCos library. The flow diagram for the application build procedure is shown in Figure 12.8.

**Figure 12.8** Application build procedure flow diagram.



As we see in Figure 12.8, we start with our C language source file `basic1.c`. In this example, we use a makefile and the GNU make utility for the build process. First, the GNU cross compiler—in our case, `i386-elf-gcc`—is invoked.

Next, the GNU linker, `i386-elf-ld`, is run, which links our application object file and the eCos library, `libtarget.a`. The other necessary eCos libraries are also linked at this time. The linker script file `target.ld`, generated during the eCos build procedure, is used during the linking process.

The output is an ELF executable file, `basic1.exe`. We use one of the GNU binary utilities, `i386-elf-objcopy`, to translate the ELF format file into an S-record file. RedBoot is used to download our application image in the S-record file.

### 12.5.1 Building the Application

The source code for this example application is located on the CD-ROM under the `examples\application` subdirectory, which includes the source code file `basic1.c` and the build file makefile. Also included in the `examples\application\bin` directory are the ELF (`basic1.exe`), S-record (`basic1.srec`), and map files for the basic application build.

Let's look at the application we are using for this example. There are two threads, A and B, that run in this application. Thread A uses a semaphore to signal Thread B. Both threads are passed a value that they print out when they run.

### STEP 1

We need to create a directory for our application code. Under our `workdir` directory, we create the subdirectory `application`. We now copy the application source code files from the CD-ROM `examples\application` subdirectory. The files we copy are `basic1.c` and `makefile`.

### STEP 2

Launch the *Cygwin Bash Shell*, if it is not already open, by double-clicking the Cygwin desktop icon or by selecting *Cygwin Bash Shell* under *Start -> Programs -> Cygnus Solutions -> Cygnus Solutions*.

From the bash shell we change to the application working directory using the command:

```
$ cd d:/workdir/application
```

### STEP 3

Next, we can build the application image. Before we build the image, let's take a moment to understand the `makefile` we are using to build our application. The `makefile` used to build this application, a portion of which is shown in Code Listing 12.3, is based on the `makefile` included in the eCos source code `examples`, located under the `examples` subdirectory in the eCos source code repository. Additional information about the make utility and makefiles can be found online at:

[www.gnu.org/manual](http://www.gnu.org/manual)

Commands and options can be passed in on the command line to build the application image as well; however, the `makefile` prevents us from having to repeatedly reenter commands. Makefiles are also useful in larger projects when multiple files are being built.

```
1 ## eCos library installation directory
2 PKG_INSTALL_DIR = /cygdrive/d/workdir/ecos/ecos_install
3
4 ## This sets the compiler to i386 PC.
5 XCC = i386-elf-gcc
6
7 ## Build flags.
8 CFLAGS = -g -Wall -I$(PKG_INSTALL_DIR)/include \
9 -ffunction-sections -fdata-sections
10 LDFLAGS = -nostartfiles -L$(PKG_INSTALL_DIR)/lib \
11 -Wl,--gc-sections -Wl,--Map -Wl,basic1.map
12 LIBS = -Ttarget.ld -nostdlib
13 LD = $(XCC)
14
15 ## Build rules.
```

```
16 all: basic1
17
18 basic1.o: basic1.c
19 $(XCC) -c -o $*.o $(CFLAGS) $<
20
21 basic1: basic1.o
22 $(LD) $(LDFLAGS) -o $@ $@.o $(LIBS)
```

**Code Listing 12.3** Example application makefile.

In Code Listing 12.3, the first part of the `makefile` sets up some variables that we can use later so that our `makefile` is easier to understand and allows repetitious use of variables without re-entering an entire string. When we want to use a variable, we use the syntax `$(VARIABLE_NAME)`. Whatever string `VARIABLE_NAME` has been declared is substituted where the variable is used.

On line 2 we set the variable `PKG_INSTALL_DIR` to the location of the eCos library that we built in the previous stage of our example. On line 5 we set the variable `XCC` to the name of the compiler we are using; in this case, it is the i386 ELF compiler (`i386-elf-gcc`).

Next, we set the build flags to different variables. Using variables allows us to substitute a variable name in place of a long string for clarity and for repeated use in multiple locations of our `makefile`.

The first variable is `CFLAGS`, shown on lines 8 and 9, which contains the flags we pass to the compiler `GCC`. The next variable is `LDFLAGS`, shown on lines 10 and 11, which contains the options for the linker `LD`. Both variables `CFLAGS` and `LDFLAGS` are located on a single line in the actual `makefile`; however, here they are split up so that all the options can be shown.

The variable `LIBS` on line 12 contains the library options that are also passed to the linker. Finally, the variable `LD` is defined on line 13 for the linker command. In this case, the linker is built into the compiler executable, `i386-elf-gcc`, so we use the `XCC` variable name to invoke the linker.

The next part of the `makefile` contains the build rules or goals. The goals are the targets that the `makefile` attempts to update. First, is `all` on line 16. This is the main target to build, which is the ELF file `basic1`.

Lines 18 and 19 contain the information to build the target `basic1.o`. As we can see on line 19, the compiler is invoked using the variable `XCC` and it is passed the flags that follow, including the string from the variable `CFLAGS`.

On lines 21 and 22 is the information to build the target `basic1`. The `basic1` target is the ELF image, which is generated using the linker and the linker options shown on line 22.

A brief description of the compiler and linker flags used is included in the `makefile`. For additional information about the various compiler flags, at the bash shell prompt you can enter the command:

```
$ i386-elf-gcc --help -v
```



The `makefile` also contains a rule, `clean`, which removes the files created during the build, such as object files. This allows us to recompile all files, not just the files modified since the last build. To clean the built files, use the command `make clean` at the bash shell prompt.

Before we proceed with the build, let's see where exactly the eCos library gets linked into the application.

```
1 STARTUP(vectors.o)
2 ENTRY(_start)
3 INPUT(extras.o)
4 GROUP(libtarget.a libgcc.a)
```

**Code Listing 12.4** Portion of `target.ld` linker script file that shows where the eCos library is included when building applications.

Line 1 of Code Listing 12.4 uses the command `STARTUP`. This causes the `vectors.o` file to be the first file linked, since the file `vectors.S` contains the entry point for our application.

The `ENTRY` command, shown on line 2, sets the entry point where the application will begin execution. In our case, the entry point is in the file `vectors.S` in the routine `_start`. Each HAL contains a file `vectors.S` that contains the startup code for that particular processor. This file is located under the `arch` source directory for each architecture.

The `INPUT` command on line 3 notifies the linker to include the named files. In this case, `extras.o` is the file, which was one of the additional files created during the eCos build procedure.

Finally, on line 4 is the `GROUP` command. This command is similar to the `INPUT` command; however, it takes archive files as its parameters. The two eCos libraries are used in this command, `libtarget.a` and `libgcc.a`. All of the files mentioned in this section of the linker script file are located in the install tree under the `ecos_install\lib` directory.

We are ready...finally!!! To build our application image we invoke the `make` utility at the bash shell command prompt using the following command:

```
$ make
```

After the build is complete, the files that are output from our build process are:

- **basic1.o**—the compiler output object file.
- **basic1.map**—the map file shows the memory layout for the image.
- **basic1.exe**—the ELF format executable file.

#### STEP 4

Next, we change the ELF file into an S-record format to load with RedBoot. For this we use the `objcopy` utility, which is one of the GNU binary utilities. To get the application image into the S-Record format we want, we use the command:

```
$ i386-elf-objcopy -O srec basic1.exe basic1.srec
```

This produces the file `basic1.srec`. This file is loaded onto the target hardware using RedBoot, as described in the next section.

### 12.5.2 Loading the Application

Now we can load and run the application using RedBoot. On the host, we use the Windows HyperTerminal program, which contains the facility to transfer files using X, Y, and Z modem protocols. As previously mentioned, RedBoot allows transfer of data using various communication methods such as serial and TFTP. Additional information about other communication methods using RedBoot can be found in Chapter 9.

One utility that can come in handy is the GNU size utility. The size utility is part of the GNU binary utilities. The size utility lists the sizes of the different sections and the total size of the object file. To get the size of our example application we run the command:

```
$ i386-elf-size basic1.exe
```

The output from the size utility is shown in Code Listing 12.5.

```
1 text data bss dec hex filename
2 76900 544 36744 113188 1ba24 basic1.exe
```

**Code Listing 12.5** Output from GNU size utility for basic application image.

In Code Listing 12.5, the section names, `text`, `data`, and `bss`, are shown on line 1. Also on line 1 is the total size of the `basic.exe` object file in decimal (`dec`) and hexadecimal (`hex`). The size values are listed under their respective sections on line 2. Adding up the `text`, `data`, and `bss` sections gives the total size of the file, 113,188 bytes, under the `dec` column, which equates to 1BA24 in hexadecimal, shown in the `hex` column.

Before continuing, ensure that RedBoot is up and running on the target hardware.

### STEP 5

To load the example application using RedBoot, we enter the command:

```
RedBoot> load -v -m yMODEM
```

The option `-v` enables verbose mode during the load process. The `-m yMODEM` option informs RedBoot that we want to use the Y modem protocol to transfer our image. After entering this command, characters (the character is C) are printed that indicate that RedBoot is waiting for the transmission of the image to begin.

To begin transferring the `basic1.srec` image, we select *Transfer -> Send File* from the menu in HyperTerminal. This brings up the dialog box shown in Figure 12.9.

As shown in Figure 12.9, we want to select Ymodem from the protocol drop-down list. Then, click the Browse button to select the `basic1.srec` file under the `D:\workdir\application` subdirectory. When the transfer begins, the status of the transfer is displayed in a dialog box similar to the one in Figure 12.10.

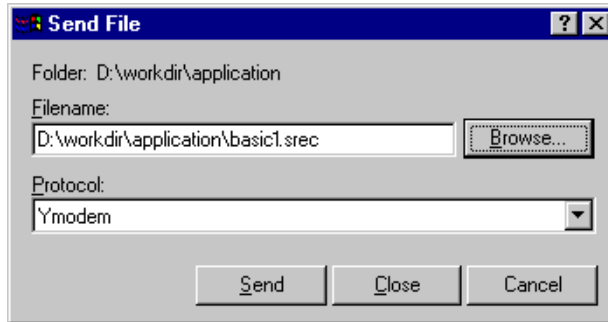


Figure 12.9 HyperTerminal Send File dialog box.

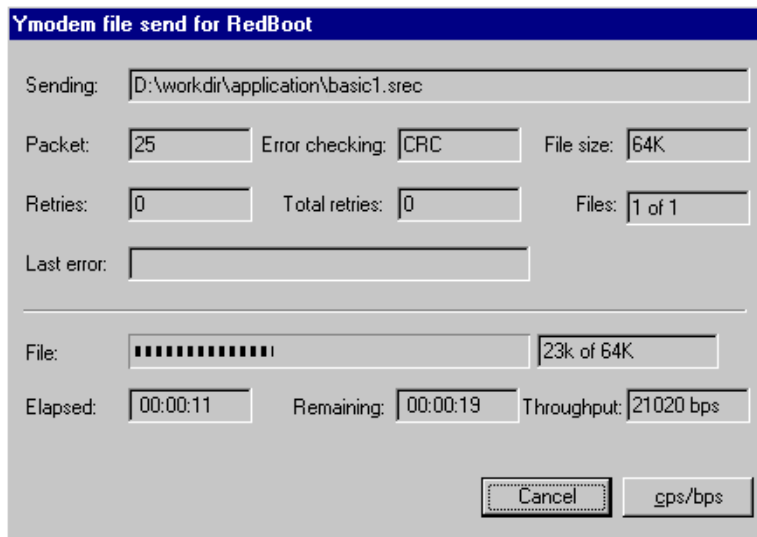


Figure 12.10 HyperTerminal Transfer Status dialog box.

The progress and information about the transfer of the application to the target hardware, including throughput and time remaining, is shown in Figure 12.10.

When complete, the dialog box is closed and RedBoot outputs a message. An example of the message output by RedBoot is shown in Code Listing 12.6.

```
1 Entry point: 0x00108000, address range: 0x00108000-0x0011aba0
2 xyzModem - CRC mode, 2(SOH)/215(STX)/0(CAN) packets, 4 retries
```

**Code Listing 12.6** RedBoot output message after transfer of our application is complete.

On line 1, in Code Listing 12.6, the memory information about the image that we just loaded is displayed. This information includes the entry point and the memory range used by the application. Line 2 summarizes the statistics of the transfer.

## STEP 6

Now we are ready to run our example application. Since RedBoot is aware of the load address for our application, we do not need to specify an entry point for the RedBoot `go` command. To run our application, at the RedBoot prompt we enter the following command:

```
RedBoot> go
```

As the `basic1` application runs, the output in Code Listing 12.7 appears on the HyperTerminal screen. The application continues to run forever.

```
1 Hello eCos World!!!
2
3 Thread A, count: 1 message: 75
4 Thread A, count: 2 message: 75
5 Thread B, message: 68
6 Thread A, count: 3 message: 75
7 Thread B, message: 68
8 Thread A, count: 4 message: 68
```

**Code Listing 12.7** Basic1 example application output.

### 12.5.3 Debugging the Application

Although RedBoot does provide some basic facilities for debugging an application, such as viewing memory locations, GDB provides more extensive debugging capabilities. Next, we are going to use GDB to connect to our target hardware, download our example application, and then run the application. Additional documentation about GDB can be found online at:

<http://sources.redhat.com/gdb>

During the host development tools setup, we built GDB and included the Insight GUI; see Chapter 10 for details. It is possible to run GDB in command-line mode, even with Insight built into GDB. Additional information about Insight can be found online at:

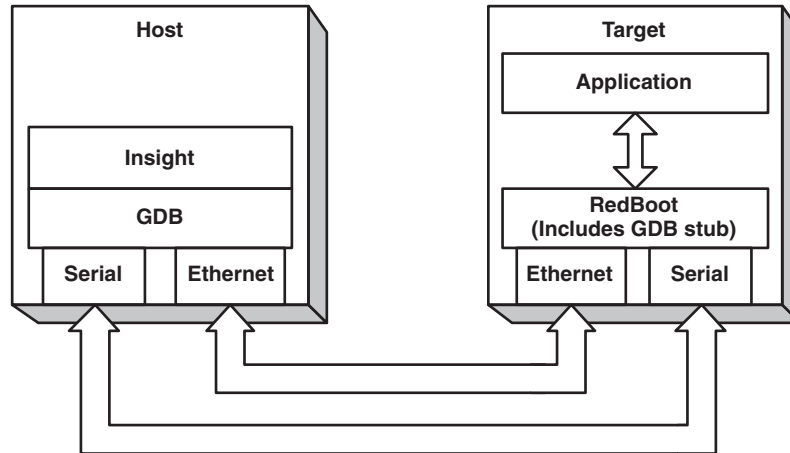
<http://sources.redhat.com/insight>

In this step, we take a brief look at using GDB in command-line mode and using the Insight graphical interface. A diagram of the debug environment is shown in Figure 12.11.

---

**NOTE** Debugging optimized code can be unreliable because the compiler might make changes in the executable file that do not exactly match the original source code. In these cases, it might be better to disable the compiler optimization when debugging source code.

To disable the compiler optimization you can either remove the `-O2` option from the Global Compiler Flags (`CYGBLD_GLOBAL_FLAGS`) or set the optimization level to zero using the option `-O0`. The eCos library and application code would then need to be rebuilt.



**Figure 12.11** The debug environment for our examples, running GDB on the host and RedBoot on the target.

As we see in Figure 12.11, GDB with the Insight GUI is run on the host development system. We can connect to our target via serial or Ethernet port. The target hardware is running RedBoot, which includes the GDB stub. The GDB stub provides the communication protocol layer for our target.

Before proceeding with the following steps, ensure that RedBoot is up and running on the target hardware.

### STEP 7

To launch GDB with the Insight GUI, we run `i386-elf-gdb.exe`. If you do not want to use the Insight GUI, you can run GDB using the no windows option; for example, `i386-elf-gdb.exe -nw`.

GDB with Insight application is shown in Figure 12.12.

Figure 12.12 shows the Insight GUI Source window in the foreground with the Console window in the background. The Console window is displayed by selecting *View -> Console*.

The first step is to load our example application. Using Insight, we select *File -> Open*, and then browse to the location of our `basic1.exe` application file.

When our example application is loaded, the source window is loaded with the file `main.cxx`.

### STEP 8

Next, we connect our host GDB to our target hardware. This is accomplished by selecting *Run -> Connect To Target*, which opens the Target Selection dialog box shown in Figure 12.13.

There are two options for connecting to the target hardware, serial or Ethernet, as shown in Figure 12.4.

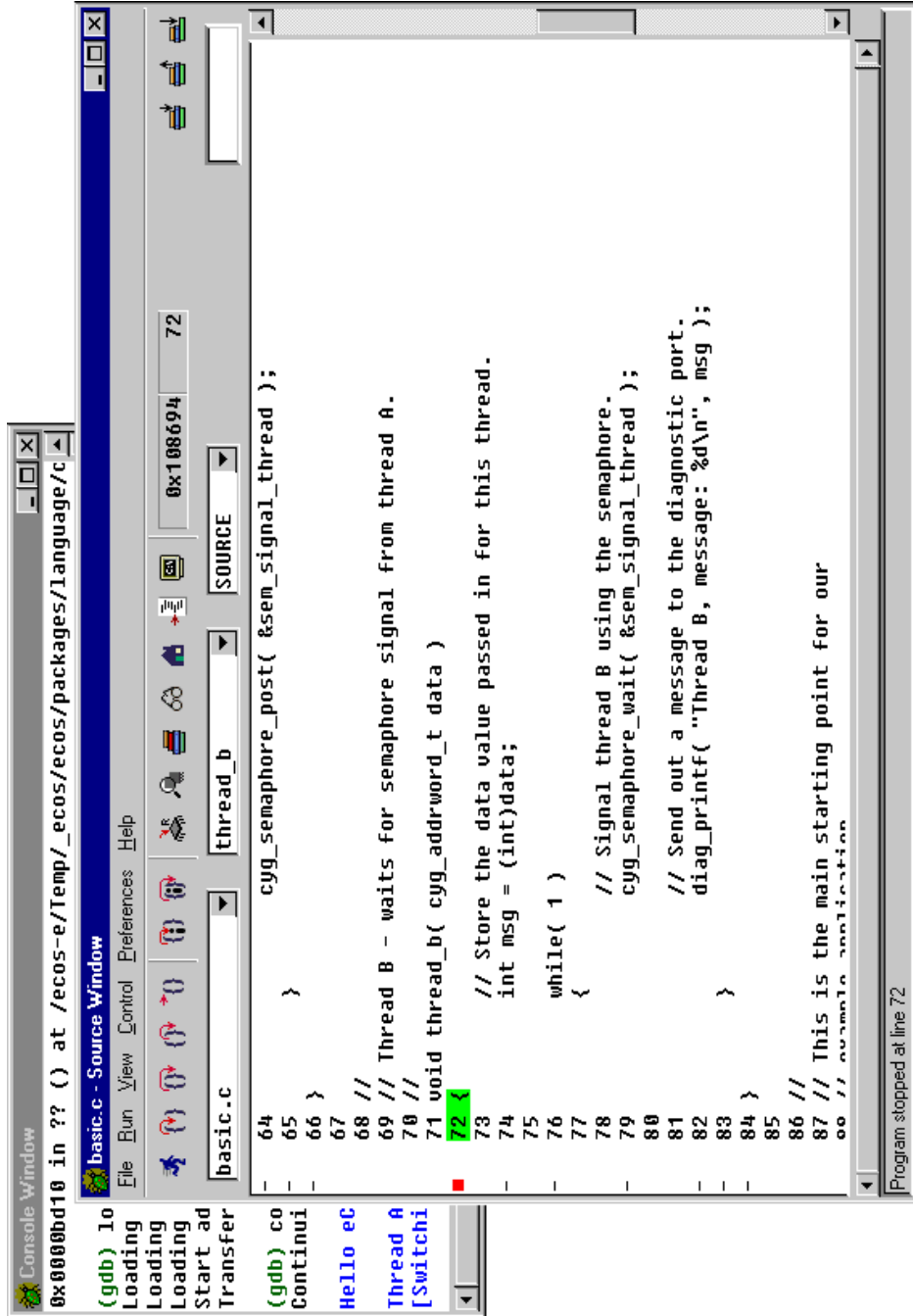
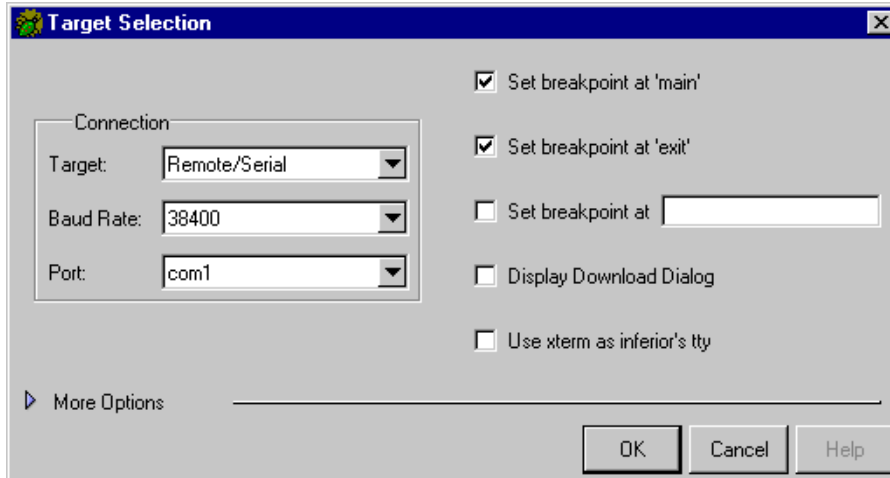


Figure 12.12 Insight GDB Source window with Console window in the background.



**Figure 12.13** Insight GDB Target Selection dialog box for connecting to the target hardware.

---

**NOTE** If connecting to the target hardware over the serial port you will need to disconnect the HyperTerminal from the serial port before attempting to connect with GDB. Both GDB and HyperTerminal cannot use COM1 simultaneously.

To connect via serial, we select *Remote/Serial* in the Target drop-down list, *38400* in the Baud Rate drop-down list, and *com1* in the Port drop-down list. The check boxes can be left in their default states. Additional connection parameters can be displayed by clicking the More Options arrow.

To connect via Ethernet, we select *Remote/TCP* in the Target drop-down list, we enter *192.168.0.10* (or whatever the RedBoot static IP address was configured as) for the Hostname, and *9000* for the Port.

After setting the connection parameters, click the OK button to connect to the target. A dialog box showing *Successfully Connected* is displayed when the GDB has connected to the target hardware.

### STEP 9

Once connected to the target, we download our example application by selecting *Run -> Download*. During the download, a progress bar appears in the upper right-hand corner of the Insight source window. The status of the download is displayed on the bottom status bar. When the application download is complete, *Download Finished* is displayed on the bottom status bar, as well as statistics about the download including elapsed time and bytes transferred.

### STEP 10

If the console window is not open, select *View -> Console*. This enables us to view the output from our application.

---

**NOTE** In some cases, the download process can be very slow using GDB and RedBoot. Some things can be done to speed up the download process. Details about speeding up an application download using GDB and RedBoot can be found in the eCos Frequently Asked Questions (FAQ), which is online at <http://sources.redhat.com/forum-serv/ecos/cache/1.html>. Another eCos FAQ is located online at <http://sources.redhat.com/ecos/faq.html>.

To start the application, select *Control* → *Continue*. The output from the `basic1` program, as shown in Code Listing 12.7, is displayed in the console window.

You can now familiarize yourself with the different functionality with GDB, such as stopping the program, setting breakpoints, and watching variables.

### 12.5.3.1 Using the GDB Command-Line Interface

As mentioned previously, GDB can also be run from the CLI if the Insight GUI is not needed. Let's go through the commands needed to load and run the application from the CLI. These commands can also be entered in the Console window when using Insight. Table 12.2 lists the CLI commands for running GDB using the serial and Ethernet ports. The commands are shown after the GDB prompt (`gdb`).

**Table 12.2** GDB CLI Commands for Serial and Ethernet Port Debugging

| Serial                                                                                              | Ethernet                                                                                                                       |
|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>STEP 1:</b> Change to the application directory.<br>(gdb) <code>cd d:/workdir/application</code> | <b>STEP 1:</b> Change to the application directory.<br>(gdb) <code>cd d:/workdir/application</code>                            |
| <b>STEP 2:</b> Set the serial port baud rate.<br>(gdb) <code>set remotebaud 38400</code>            | <b>STEP 2:</b> Connect to the target via the Ethernet port. <sup>a</sup><br>(gdb) <code>target remote 192.168.0.10:9000</code> |
| <b>STEP 3:</b> Connect to the target via the serial port.<br>(gdb) <code>target remote com1</code>  | <b>STEP 3:</b> Load the application<br>(gdb) <code>load basic1.exe</code>                                                      |
| <b>STEP 4:</b> Load the application.<br>(gdb) <code>load basic1.exe</code>                          | <b>STEP 4:</b> Run the application.<br>(gdb) <code>continue</code>                                                             |
| <b>STEP 5:</b> Run the application.<br>(gdb) <code>continue</code>                                  |                                                                                                                                |

<sup>a</sup> Use the static IP address configured when RedBoot was built. The port number, 9000, is entered following the IP address.



## 12.6 The eCos Tests

The eCos repository provides test suites for various packages. The tests are designed to thoroughly exercise the various functionality within the different packages to ensure proper operation. These tests exercise the software at the module, component, and system levels and include stress and performance testing.

The eCos repository includes test suites for the kernel, various device drivers, and the I/O Sub-System. The kernel suite includes tests that exercise the different synchronization mechanisms, run various thread-related functions, and use the different clock functionality. The I/O Sub-System includes various serial port tests. There are also various tests for the different target hardware platforms.

The test source code is contained in the eCos source code repository and located under the `tests` directory within a package's directory structure; see Figure 11.1 for an illustration of the generic package directory structure. The test source code also provides a great example of how to implement various functionality within an application. These tests can be used to guide you through understanding the different eCos APIs.

Now let's focus on how we can use the Configuration Tool to build and run various tests in the eCos framework. The tests are built using the Configuration Tool by selecting *Build -> Tests*. The output files from the test build procedure are located in the install tree under the `tests` directory. The output files are ELF format files.

---

**NOTE** Some tests, such as those included in the Basic Networking Framework package, contain a configuration option that must be enabled to build the specified tests. The *Build Networking Tests (Demo Programs)* (`CYGPKG_NET_BUILD_TESTS`) configuration option, when enabled, allows additional networking tests to be built.

After the tests are built, the Configuration Tool also facilitates automatically downloading and running the tests on the target hardware. To run the tests using the Configuration Tool, select *Tools -> Run Tests*. This brings up the Run Tests dialog box as shown in Figure 12.14.

The first tab, Executables, displays all of the tests that are available for running on the target hardware. Checking the box next to the test selects the test for running. Tests can be added or removed using the buttons at the top of the dialog box. Once the tests are configured to run, the Run button at the bottom of the dialog box begins execution of the selected tests.

The next tab is Output. This displays output from the tests as they execute on the target hardware. The Summary tab keeps track of the results of each test, including the time it took to execute and the status as to whether the test passed or failed.

Prior to running the tests, the method for connecting to the target hardware is selected. Clicking the Properties button at the bottom of the Run Tests dialog box brings up the Settings dialog box as shown in Figure 12.15.

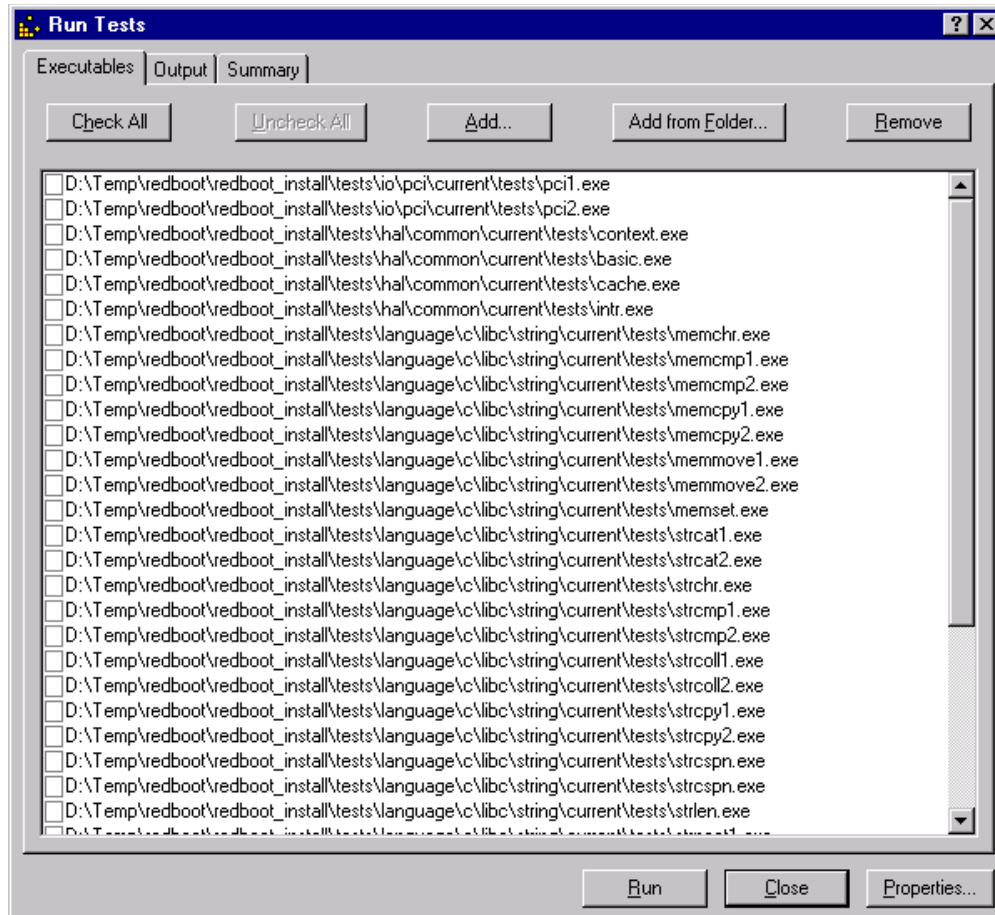


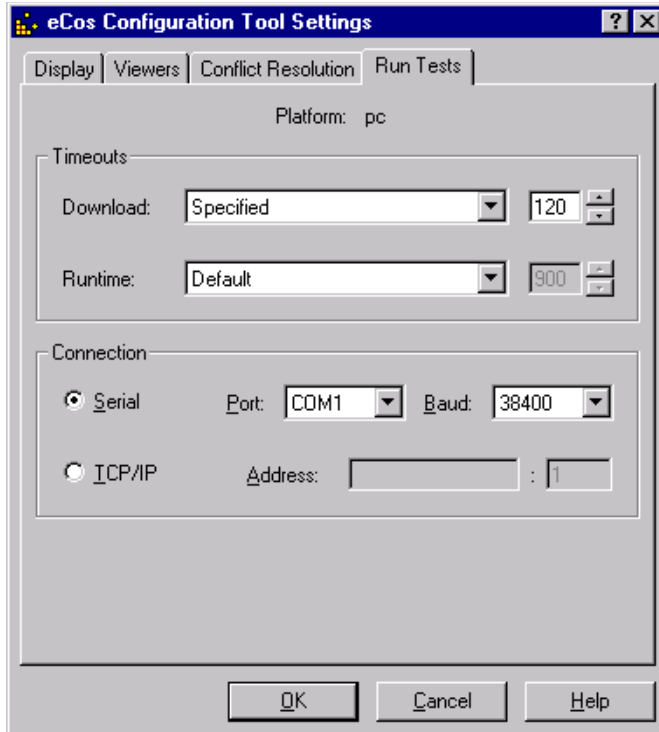
Figure 12.14 Configuration Tool Run Tests dialog box.

As we see in Figure 12.15, the timeout period can be specified. The connection to the target hardware can also be set to either serial or Ethernet.

## 12.7 Simulators

The eCos framework provides simulators for several different processor architectures, including the Hitachi H8/300, MIPS, Matsushita AM3x, PowerPC, and SPARClite. A simulator can be useful when a development board is not available, or possibly too costly, and the hardware has not been developed. In these cases, the software can be developed for the target processor and run on a simulator target.

The simulators are run from the GNU debugger (GDB), either using the Insight GUI or from the GDB command line. To invoke the PowerPC-based GNU debugger you run the executable `powerpc-eabi-gdb.exe`.



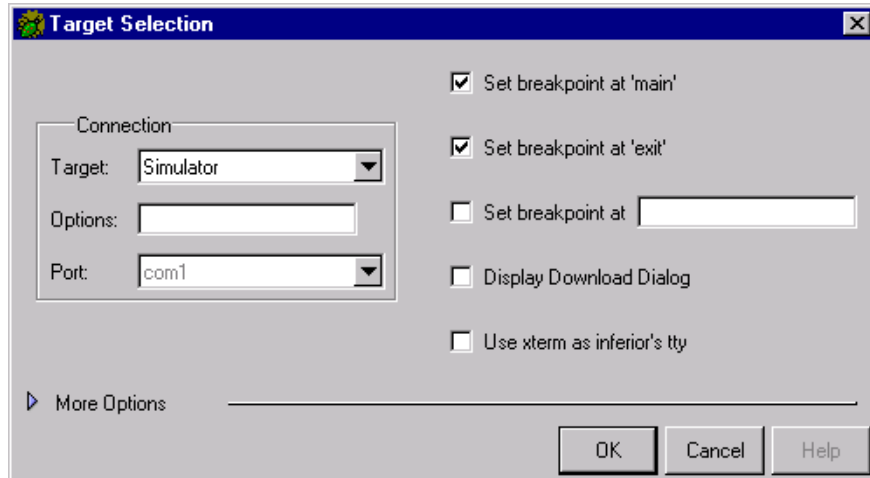
**Figure 12.15** Configuration Tool Connection Settings dialog box for running tests on a target system.

**NOTE** The pre-built PowerPC GNU cross-development tools are also included on the CD-ROM in the file `ppcgnu-tools.tar.bz2` under the `gnu\ppctools` directory. You can add the PowerPC GNU cross-development tools by unzipping the file under the root `D:\cygwin` directory. The files are extracted under the `D:\cygwin\toolsppc` directory. The command to extract the PowerPC GNU cross-development tools is:

```
$ tar xjvf /cygdrive/e/gnu/ppctools/ppcgnutools.tar.bz2
```

You would then need to add the `D:\cygwin\toolsppc` directory to your path, as shown in Chapter 10 in Section 10.2.2, *Installing the Platform-Specific Cross-Development Tools*, in STEP 4. The PowerPC GNU cross-development tools are used in Chapter 13.

First, let's see how to run code on the simulator using Insight. When running Insight GDB, select *File* → *Target Settings* to bring up the dialog box for connecting to a target, as shown in Figure 12.16. Under the Target drop-down list, select Simulator. Any target-specific options can be entered in the Options edit field within this dialog box.



**Figure 12.16** PowerPC-based Insight GDB Target Selection dialog box.

Using the GDB command line, we use the following commands, assuming we are running a program named `ecoshello.exe`:

First, load the symbols from the file with the command:

```
(gdb) file ecoshello.exe
```

Next, set our target to the simulator:

```
(gdb) target sim
```

On successful connection to the simulator, the output message `Connected to simulator` is displayed. If there are any target-specific options, they can be entered at the end of the command. Then, we load the program using the command:

```
(gdb) load
```

Finally, the program is run on the simulator with the command:

```
(gdb) run
```

The program is debugged using the standard GDB commands. Using a simulator allows some of the software to proceed without hardware being present; however, certain software modules, such as device drivers, rely on the presence of the target hardware.

## 12.8 Summary

In this chapter, we got down to the practical aspects of using eCos by running through some examples. We started with an overview of the eCos build process, which we use for generating eCos libraries or RedBoot images.

After covering the details of the build procedure, we began our examples with a build and install of the RedBoot ROM monitor. This provided us with a method for downloading

our application code onto our target hardware, as well as providing GDB debugging support on the target.

We then proceeded to construct an eCos library from our configuration. The eCos library was used to build our application. Next, we focused on loading our application using first Red-Boot directly and then using GDB for debugging.

We concluded this chapter with a brief look at the eCos test suite and the simulators provided in the eCos framework. Although these examples were basic, they are good starting points for understanding the eCos development environment and provide a baseline for extending the functionality to meet your own requirements.

These examples enable you to evaluate the eCos real-time operating system and become familiar with the build and debug tools available. Moving on to more complex examples by incorporating additional features into the eCos configuration is a good next step.

## Porting eCos

**G**etting your application running on your new target hardware platform is typically the main goal in embedded software development. This typically includes porting eCos to the target hardware as well. eCos was designed for portability by using a layered approach to the different software system components. When porting eCos, moving the HAL to the new target hardware is the first step, which enables the higher layer components, such as the kernel, to also run on the target hardware. Additional functionality, such as display drivers and Ethernet drivers, can then be incorporated to meet the requirements of the system and get the system fully operational.

Because of the unique nature of each new target hardware system, it is impossible to cover every possible detail for each target system. Instead, in this chapter, we look at a platform porting example to get an overall understanding of the eCos porting process. Using one of the supported evaluation platforms as a baseline, we can add a new target platform to the eCos framework and see how to get this new platform up and running using the eCos tools. We can then go through some of the details that might need to be addressed in other porting procedures.

The PowerPC GNU cross-development tools are included on the CD-ROM in case you want to build this porting example, although this is not necessary for understanding the porting process. These tools are located under the `gnu\ppctools` directory.

### 13.1 Overview of Porting

Now that we have an understanding of the components in the eCos system, how to use the tools, and how to write an application using eCos, we can focus on the main task involved with using any RTOS: getting the software running on our own hardware. This is typically the main goal

when using an RTOS. Porting to a new hardware platform can be a painful and slow process because it is typically occurring at the same time that the hardware itself is being debugged.

The layered architecture of eCos, as described in Chapter 1, *An Introduction to the eCos World* and shown in Figure 1.1, makes the porting process a bit easier. To move to a new hardware platform we start by porting the HAL and ensure that it functions properly on the new hardware. Since the higher application layers sit on top of the HAL, they are not dependent on a specific piece of hardware. Options can then be configured to meet the needs of the new hardware platform.

Since every piece of hardware is unique in its own way, the porting process described in this chapter should be used as a general guide. Your hardware might require its own specific code such as serial and display drivers to get the system operating properly. There will be implementation decisions that you need to make for your hardware in order to port eCos.

The porting of the eCos HAL can be broken down into three different types: Platform, Variant, and Architecture porting. Platform porting consists of using the HAL of a current platform supported by eCos, which is similar to the new hardware platform, as a baseline and then making modifications for the new platform, including the new memory layout and any specific initialization needed. New drivers might need to be developed to accommodate additional hardware resources. Of the three porting types, platform porting usually requires the least amount of effort if an existing HAL can be used as a baseline.

Variant porting also uses a closely related existing HAL as a baseline. As described in Chapter 2, *The Hardware Abstraction Layer*, a variant HAL supports differences of a specific processor from the generic processor architecture. A variant port might consist of redefining interrupts, cache, or other features that override the default implementation of the architecture HAL. A variant port requires an existing architecture HAL port. The amount of effort necessary for a variant port directly corresponds to the similarity of the existing variants for the specific processor.

Finally is architecture porting. This consists of using an existing architecture HAL as a baseline. The reason why an architecture port can be a daunting task is that compiler support must be present before beginning the porting process. eCos is closely coupled to the GNU C/C++ Compiler. Although the GNU C/C++ Compiler supports numerous different architectures, if it does not support your new architecture then the task of porting the GNU compiler needs to be completed as well. After the architecture HAL port is complete, a platform port must be done in order to support the new hardware.

Before setting out on a new variant or architecture port it is a good idea to search the eCos discussion mailing list archives to see if anyone else in the development community is currently undertaking the same port. You can then see if the developer intends to contribute the port back for use by others. The eCos discussion mailing list archives can be searched online at:

<http://sources.redhat.com/ml/ecos-discuss>

If no one else is performing the same port, you can post your intentions to the list using the address `ecos-discuss@sources.redhat.com`. You might receive a reply giving you some additional tips or resources to use to help you out.

## 13.2 A Platform Porting Example

In order to get a better understanding of the platform porting procedure, we are going to go through the process of creating a new hardware platform for our imaginary piece of hardware. In this example, we use the Motorola MBX860 platform HAL as our baseline. The Motorola MBX860 HAL is located under the `hal\powerpc\mbx` subdirectory.

During the software development process, choosing an off-the-shelf evaluation board allows us to do parallel software development while the hardware is being designed. Although the specific platform implementation details might differ among various HALs, this example should be used as a general guideline that can be applied to all platform ports.

For this porting example, we use RedBoot as the template. Additional information about RedBoot can be found in Chapter 9, *The RedBoot ROM Monitor*. Since RedBoot is essentially a thin command-line interface sitting on top of the HAL with GDB stub functionality included, we can leverage off the RedBoot development to put us steps ahead in the porting process.

Once the HAL is ported to the new hardware, RedBoot gives us the ability to load and debug code on our new target platform. Initially, we want to strip out any unnecessary functionality included in the RedBoot template so that we have a “bare bones” build of the RedBoot image. As we progress with the port to our new hardware, we can add back the functionality we need to produce a full-featured RedBoot image.

It is important to establish some baseline hardware functionality prior to setting out on the eCos port. We should wring out any basic hardware problems, such as memory reading or writing errors, before proceeding. This baseline functionality can be established using some basic RAM read/write tests and ROM, if a flash ROM device is used, read/write tests. Having the proper tools during the early stages of bringing up hardware is crucial. Most processors have some type of In-circuit Emulator (ICE) or Background Debug Mode (BDM) tools available. A good article about ICE technology can be found online at:

[www.embedded.com/1999/9910/9910sr.htm](http://www.embedded.com/1999/9910/9910sr.htm)

Having one of these tools can give you the ability to load code onto the target hardware without going through the process of burning EPROMs whenever you need to test code changes. Easing the testing of new code changes is essential during the eCos porting process.

Other equipment that is good to have when bringing up a new piece of hardware is a logic analyzer and oscilloscope. These pieces of equipment can guarantee that the hardware is meeting their associated timing specifications.

An evaluation board is a great piece of hardware to have during the initial phases of a project. Using the evaluation board, you can begin writing the application software for your system long before your own hardware is ready. Then, when your new hardware comes online you can go through the platform porting procedure, as described later in this section, to get eCos up and running on your new platform. The evaluation board also gives you a baseline of functionality that you can use to check your software as the project progresses. If something is not working correctly on your new hardware, you can always test it out on the evaluation platform to see how

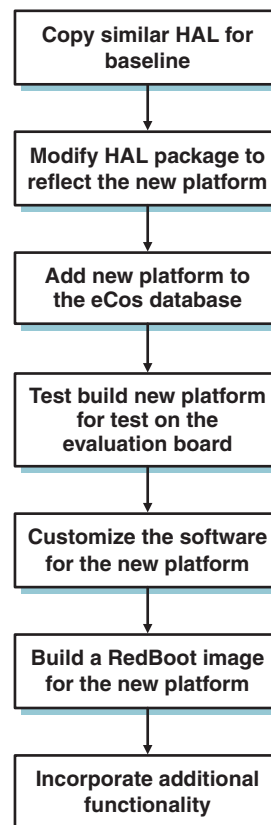


it works in a more stable environment. A list of the evaluation boards supported by eCos can be found in Appendix A, *Supported Processors and Evaluation Platforms*. The latest additions to this list can also be found online at:

<http://sources.redhat.com/ecos/hardware.html>

Figure 13.1 is an overview flowchart of the platform porting example we go through in this chapter.

**Figure 13.1** eCos porting process flowchart.



As we see in Figure 13.1, we begin by basing our new platform on an existing evaluation board HAL package. This allows us to quickly set up the files we need to get our new platform package noticed by the eCos framework. Then we modify the package to make it specific for our new hardware platform.

Next, we add the new platform package to the eCos database so that we can use the configuration tools to build our new platform image. At this point, we do a test build, which we run on our evaluation board, to ensure that the package is built properly.

Now we customize the software for our new target platform, including changing clock configuration, memory layout, and adding or modifying device drivers to match the new hardware. Then we are able to build a RedBoot image for our new platform. Once this RedBoot image is up and running, we are able to extend the functionality for our new platform by incorporating any additional features required.

For our example, we are performing the platform port to our new hardware that we code name “*Martini*”. The example code for our new *Martini* HAL package is located on the CD-ROM under the `examples\martini` directory. The *Martini* package is contained under the `examples\martini\package` directory and the install tree, MLT directory, and eCos configuration file are located under the `examples\martini\build` directory. The modified eCos database file `ecos.db`, which shows the modifications necessary to get our new *Martini* platform noticed in the eCos framework, is also included on the CD-ROM.

---

**NOTE** One of the steps in this example porting procedure entails building a RedBoot image for the new PowerPC-based target platform. The PowerPC GNU cross-development tools binary files are included on the CD-ROM if you would like to build the *Martini* example image.

To set up the PowerPC GNU cross-development tools follow these steps.

**STEP 1:**

Open a bash command shell. Change to the root Cygwin directory by entering the command:

```
$ cd /
```

**STEP 2:**

Unzip the PowerPC GNU cross-development tools with the command:

```
$ tar xjvf /cygdrive/e/gnu/ppctools/ppcgnutools.tar.bz2
```

After executing this command, the PowerPC GNU cross-development tools are located under the `D:\cygwin\toolsppc` directory. The binary executables are under the `D:\cygwin\toolsppc\H-i686-pc-cygwin\bin` directory.

**STEP 3:**

Next, we set the path for our new GNU cross-development tools. The bash shell command for this is:

```
$ PATH=/toolsppc/H-i686-pc-cygwin/bin:$PATH ; export PATH
```

Building the *Martini* image using the PowerPC GNU cross-development tools is not necessary in order to understand the eCos porting procedure.

### 13.2.1 PowerPC HAL Directory and File Structure

Let's take a closer look at the PowerPC HAL directory structure focusing on the Motorola MBX860 platform. We need to know the location of the files and subdirectories that contain the functionality for the MBX platform. Figure 13.2 shows the relevant HAL directories for the Motorola MBX860 platform. The version subdirectories are left out in Figure 13.2; we are using the current version in our example. Other HAL platforms have similar directory structures.

The subdirectory `common` contains the package configuration files general to all HAL architectures, including files for general interrupt configuration, virtual vector layout, and HAL debugging control. Function wrappers are contained in this subdirectory to create the commonality found among all HAL implementations.

The `arch` subdirectory is located under the `powerpc` architecture HAL directory and includes files for generic support for the PowerPC processor architecture. Functionality included in this generic support consists of exception vector initialization, ROM and RAM startup configuration, common interrupt and exception handling, thread context switch handling, a generic linker script file, and common debugging functions.

The `mbx` subdirectory contains the Motorola MBX860 platform source files. The description, in CDL script format, of the MBX package is in the file `hal_powerpc_mbx.cdl` located in the `cdl` subdirectory.

The `include` subdirectory contains all header files for the MBX platform. Under the `include` subdirectory is the `pkgconf` subdirectory, which is where the memory layout files are located.

The `misc` subdirectory includes RedBoot minimum configuration files that can be imported into the Configuration Tool to establish a baseline of configured options and packages.

The subdirectory `src` contains the main source assembly file, `mbx.S`, for the MBX platform. This file contains the hardware setup code in the routine `hal_hardware_init`, which configures all necessary processor registers, and initializes the chip select configuration for the MBX platform. The `tests` subdirectory contains tests for the MBX platform.

The `mpc8xx` and `quicc` subdirectories are shown because they are variants of the MPC860 architecture used by the MBX platform. We need to be aware of the variants used by the platform we are using as a baseline in case we need to modify or extend the support offered in the variant packages.

As we proceed with the platform port, we are going to also need to be aware of the driver packages that are included in the MBX platform package. These might also need to be modified to support the new hardware.

#### **STEP 1** Getting the New Platform Noticed

The first step in the platform porting process is to get our new hardware platform noticed by the Configuration Tool, which allows us to build the software for our new hardware platform.

We first create a `martini` directory under the PowerPC HAL directory because our hardware uses the PowerPC MPC860T processor.

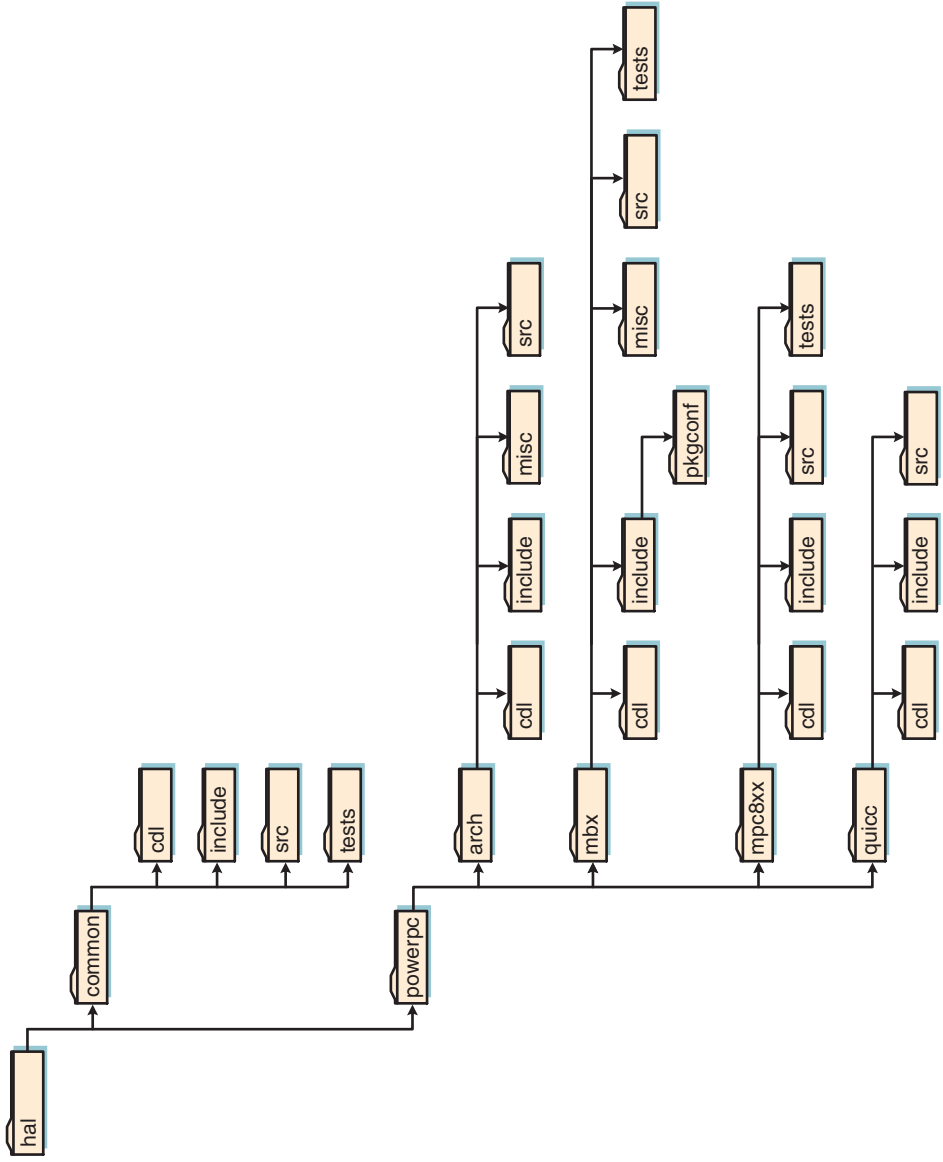


Figure 13.2 Motorola MBX860 Platform HAL directory structure.

Since we are using the MBX platform as our baseline, we copy the subdirectory `hal\powerpc\mbx\current` into our new hardware subdirectory `hal\powerpc\martini\current`.

---

**NOTE** If the files under our new hardware platform are read-only, it is easier to make these files read/write so we can make proper modifications. When using the *Martini* example from the CD-ROM, the files are marked as read-only. When you copy them to your working directory, be sure to mark them read/write as necessary.

Next, we need to change the filenames from the MBX platform to represent our *Martini* platform. Table 13.1 shows an example of the filename changes needed for our *Martini* package, which are made to the files in our new subdirectory `hal\powerpc\martini\current`. The subdirectory location of the files is given in Table 13.1 as well.

**Table 13.1** HAL Platform Porting Filename Changes

| <b>MBX Platform Filename</b>                     | <b><i>Martini</i> Platform Filename</b>  |
|--------------------------------------------------|------------------------------------------|
| <i>Subdirectory</i> <code>cdl</code>             |                                          |
| <code>hal_powerpc_mbx.cdl</code>                 | <code>hal_powerpc_martini.cdl</code>     |
| <i>Subdirectory</i> <code>include\pkgconf</code> |                                          |
| <code>mlt_powerpc_mbx_ram.h</code>               | <code>mlt_powerpc_martini_ram.h</code>   |
| <code>mlt_powerpc_mbx_ram.ldi</code>             | <code>mlt_powerpc_martini_ram.ldi</code> |
| <code>mlt_powerpc_mbx_ram.mlt</code>             | <code>mlt_powerpc_martini_ram.mlt</code> |
| <code>mlt_powerpc_mbx_rom.h</code>               | <code>mlt_powerpc_martini_rom.h</code>   |
| <code>mlt_powerpc_mbx_rom.ldi</code>             | <code>mlt_powerpc_martini_rom.ldi</code> |
| <code>mlt_powerpc_mbx_rom.mlt</code>             | <code>mlt_powerpc_martini_rom.mlt</code> |
| <i>Subdirectory</i> <code>src</code>             |                                          |
| <code>mbx.S<sup>a</sup></code>                   | <code>martini.S<sup>a</sup></code>       |

<sup>a</sup> In this case, it is important that the extension of the assembly file is an uppercase “S” rather than a lowercase “s”. The uppercase “S” allows the GNU C compiler driver to perform preprocessing on the assembly file.

Now we need to modify our new CDL script file, `hal_powerpc_martini.cdl` located in the `hal\powerpc\martini\current\cdl` subdirectory, to reflect our *Martini* hardware platform. The modifications necessary for the *Martini* hardware platform consist of changing references to the MBX board to our new *Martini* platform, which can be done by a search and replace.

Code Listing 13.1 shows an example of the changes made for the new *Martini* package. The modified file for the *Martini* platform, `hal_powerpc_martini.cdl`, is located on the CD-ROM in the `examples\martini\package\current\cdl` subdirectory.

```
1 cdl_package CYGPKG_HAL_POWERPC_MARTINI {
2 display "Martini PowerPC board"
3 parent CYGPKG_HAL_POWERPC
4 requires CYGPKG_HAL_POWERPC_MPC8xx
5 define_header hal_powerpc_martini.h
6 include_dir cyg/hal
7 description "
8 The MARTINI HAL package provides the support
9 needed to run eCos on a Martini board
10 equipped with a PowerPC processor."
11
12 compile hal_diag.c hal_aux.c martini.S
13
14 implements CYGINT_HAL_DEBUG_GDB_STUBS
15 implements CYGINT_HAL_DEBUG_GDB_STUBS_BREAK
16 implements CYGINT_HAL_VIRTUAL_VECTOR_SUPPORT
17 .
18 .
19 .
20 }
```

**Code Listing 13.1** Example CDL script file changes, in the file `hal_powerpc_martini.cdl`, for the new *Martini* platform.

As we see in line 1, we change the `cdl_package` command name from `CYGPKG_HAL_POWERPC_MBX` to `CYGPKG_HAL_POWERPC_MARTINI`. We also change the CDL command `display` description on line 2. You can choose whatever descriptive name you wish for the `display` and `description` CDL commands.

Lines 3 and 4 are left unchanged at this point in the porting process. We will revisit these values later in the porting process. Line 5 needs to reflect our new header file `hal_powerpc_martini.h`. Lines 7 through 10 contain the new description for our *Martini* platform. On line 12, for the CDL command `compile` we change the file `mbx.S` to `martini.S`. We might need to add files to this statement later if necessary. The remaining lines 13 through 16 remain the same as those in the MBX CDL script file.

There are locations in two source files that we need to change for our new *Martini* platform. The first is located in the file `plf_stub.h` under the `include` subdirectory. We need to change the line:

```
#include <pkgconf/hal_powerpc_mbx.h>
to
#include <pkgconf/hal_powerpc_martini.h>
```

We need to perform this same modification to the file `martini.S` under the `src` subdirectory. It can be useful to use the `grep` utility, to find the `mbx` string, during this modification step.

---

**NOTE** You might find that additional modifications are needed to various source files for your new hardware if a different HAL platform is used as a baseline.

Next, we need to get our *Martini* platform noticed by the Configuration Tool. To do this, we add our new platform package to the `ecos.db` file located under the `D:\ecos\packages` subdirectory.

---

**NOTE** It is a good idea to make a backup copy of the original `ecos.db` file before editing the file, just in case you need to refer to it or restore the original file.

As we know from Chapter 11, *The eCos Toolset*, the `ecos.db` file uses the CDL and contains the high-level descriptions for all packages in the component framework. Code Listing 13.2 shows the CDL description additions to the `ecos.db` file in order to get the Configuration Tool to recognize our *Martini* platform.

```
1 package CYGPKG_HAL_POWERPC_MARTINI {
2 alias { "Martini board" hal_powerpc_martini powerpc_martini_hal }
3 directory hal/powerpc/martini
4 script hal_powerpc_martini.cdl
5 hardware
6 description "
7 The MARTINI HAL package provides the support
8 needed to run eCos on a Martini board equipped
9 with a PowerPC processor."
10 }
11 .
12 .
13 .
14 target martini {
15 alias { "Martini board" martini860 }
16 packages { CYGPKG_HAL_POWERPC
```

```
17 CYGPKG_HAL_POWERPC_MPC8xx
18 CYGPKG_HAL_POWERPC_MARTINI
19 CYGPKG_HAL_QUICC
20 CYGPKG_IO_SERIAL_POWERPC_QUICC_SMC
21 CYGPKG_DEVS_ETH_POWERPC_QUICC
22 CYGPKG_DEVS_FLASH_MBX
23 CYGPKG_DEVS_FLASH_AMD_AM29XXXXX
24 }
25 description "
26 The martini target provides the packages needed
27 to run eCos on a Martini board."
28 }
```

**Code Listing 13.2** eCos repository database file, `ecos.db`, changes for the *Martini* platform.

The easiest way to add the proper package and template descriptions to the `ecos.db` file is to copy the package and template descriptions from the HAL platform we are using as a baseline. In our case, this is the MBX package and MBX template. We then change the MBX names to our new *Martini* platform name.

The package description and template description are in different locations in the `ecos.db` file. We want to maintain this structure by copying the package and template descriptions in the same area as the MBX platform we are using as a baseline. For additional information about how the `ecos.db` file descriptions are reflected in the Configuration Tool user interface, see Chapter 11.

On line 1, we change the package CDL command from `CYGPKG_HAL_POWERPC_MBX` to `CYGPKG_HAL_POWERPC_MARTINI`. As we can see, this is the same package name used in the *Martini* CDL script file shown in Code Listing 13.1.

We also change the MBX references made on lines 2 through 4 *Martini*. Line 4 informs the Configuration Tool where to find the CDL script file to describe our new *Martini* package—in our case, the filename is `hal_powerpc_martini.cdl`—which we previously added and edited. The `description` CDL command is changed to depict the *Martini* package.

Lines 11 through 13 are there to remind us that the package CDL command and the target CDL command are in different locations in the `ecos.db` file.

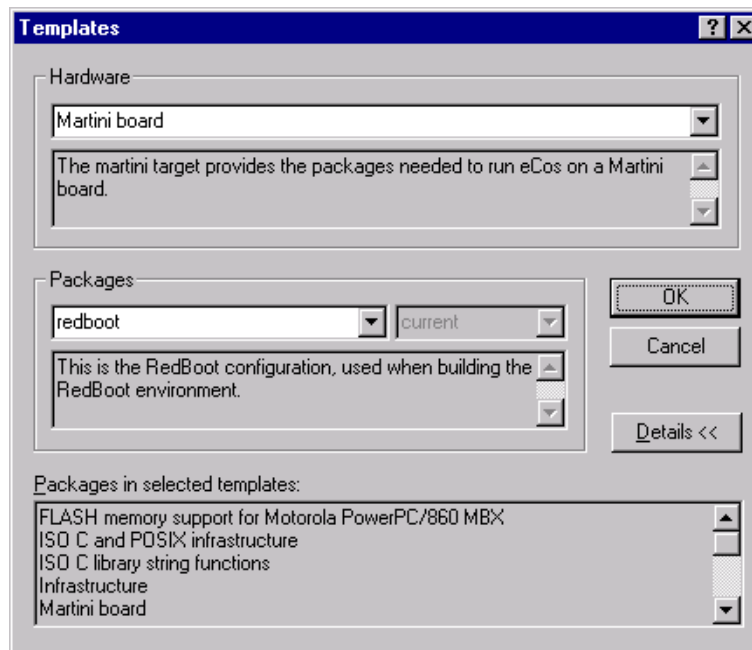
Finally, we change the CDL command `target` for our new platform. Line 14 shows our new target name `martini`. The `alias` CDL command is also changed for the *Martini* platform. The CDL command `packages`, shown on lines 16 through 24, is used by the Configuration Tool to determine which packages to load for our *Martini* platform. At this point, we only want to change the package name `CYGPKG_HAL_POWERPC_MBX` to `CYGPKG_HAL_POWERPC_MARTINI` on line 18. At this point, we do not change the `CYGPKG_DEVS_FLASH_MBX` package on line 22; we modify the packages needed to build our new platform later in the porting process. Finally, change the `description` CDL command to give an explanation our new *Martini* platform.



## STEP 2 Test Build of the New Platform

What we have now in our example port is a new platform named *Martini* that contains all of the functionality to operate the Motorola MBX860 evaluation board. This next step verifies the changes we made to the CDL script files and allows us to build a RedBoot image with the *Martini* name that has the functionality to operate the MBX evaluation board. This allows us to verify the build process using our new hardware package.

To verify that our new *Martini* package is present, first we need to launch the Configuration Tool. In the Configuration Tool, we can select our new *Martini* package using the Templates dialog box. The Templates dialog box is displayed by selecting *Build -> Templates* from the menu. Figure 13.3 shows our new *Martini* template selected from the hardware platform drop-down list.



**Figure 13.3** Templates dialog box showing our new *Martini* platform package.

As we see in Figure 13.3, the modified description we made to the `ecos.db` file is displayed under the *Martini Board* hardware platform selection. We also want to select *redboot* from the Packages drop-down list. Click the OK button to load our new *Martini* package.

---

**NOTE** The Resolve Conflicts dialog box might pop up after selecting the new template. This occurs because we are changing from the default configuration loaded when the Configuration Tool loads to our new *Martini* platform configuration. If the Resolve Conflicts dialog box is displayed, click the Continue button to proceed with resolving all conflicts.

Figure 13.4 shows our new *Martini* loaded into the Configuration Tool. The Configuration window shows our new *Martini PowerPC Board* package selected. The Properties window displays the information from the CDL script file, `hal_powerpc_martini.cdl`, that we modified in step 1. The Short Description window also shows our description of the new *Martini* package.

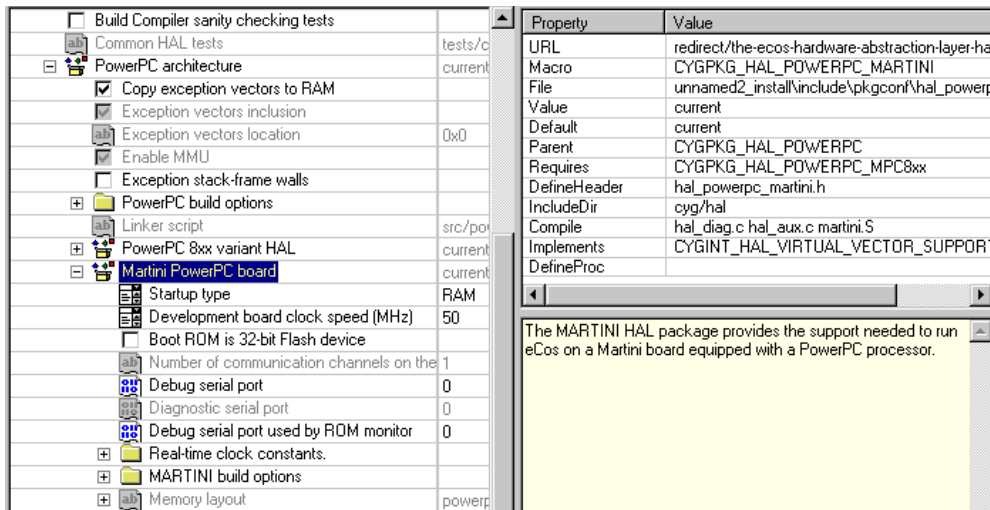


Figure 13.4 The Configuration Tool interface showing our new *Martini* package.

Now that we have our new *Martini* platform loaded with the functionality to run the MBX board, we can test the build process. The output of this build generates a RedBoot image that we can load onto our MBX board to verify the build process. Being able to verify the image we build on an evaluation board is a good reason to have a stable hardware platform to use during the porting process.

Before building the RedBoot image, we want to make sure that all configuration options are set properly for the evaluation board. In our case, since we are using the MBX development board, we want to select ROM for the *Startup Type* configuration option under the *Martini PowerPC Board* package. We also need to verify that the other HAL options, such as *Development Board Clock Speed*, are configured for the MBX evaluation board.

**NOTE** If you installed the PowerPC GNU cross-development tools in order to build the Martini porting example image, you will need to change the location of the build tools that the Configuration Tool uses. Select *Tools* → *Paths* → *Build Tools* from the menu. Next, browse to the `D:\cygwin\ppctools\H-i686-pc-cygwin\bin` directory and click the OK button.

After the necessary changes have been made, we can save our configuration changes for our new *Martini* platform using the *File* → *Save As* menu bar item. We use our working directory, `workdir\martini`, to store the porting project for our new *Martini* platform and use the

filename `martini.ecc`. The `martini.ecc` file used in this example, along with the install tree and MLT directory, are included on the CD-ROM for reference under the `examples\martini\build` directory.

Now we can select *Build* → *Library* from the menu bar to start the build process. It might be helpful to refer to Chapter 12, *An Example Application Using eCos*, for additional information on the build process.

---

**NOTE** You might find that additional modifications are needed to various source files for your new hardware if a different HAL platform is used as a baseline. Error messages that occur during the RedBoot image build process are displayed in the Configuration Tool output window.

When the build completes successfully, the final RedBoot image is located in the subdirectory `martini_install\bin`. We can then use the appropriate file format to load our new RedBoot image onto the evaluation board to verify that it operates properly, as shown in Chapter 12. It might be necessary to use the GNU Binary utility `objcopy` to convert the file format.

If RedBoot runs normally (basically, if we see the RedBoot prompt), we know that we have successfully added the new *Martini* platform to our local eCos repository. We can now move on with the porting procedure.

### STEP 3 Customize the New Platform Package

The next step is to customize the packages and configuration options for our new hardware target. This consists of going through each of the CDL commands in the CDL script files associated with our new hardware platform to make modifications and additions that match the new hardware.

There might be additional CDL script files that we need to modify other than the ones we edited in step 1. The extent of the changes necessary depends on the HAL package used for a baseline and the differences between the baseline HAL and the new hardware platform HAL. In general, we need to check the real-time clock/counter configurations and the device driver packages included, specifically for serial ports and flash memory devices.

---

**NOTE** It is best to eliminate as many of the packages as possible and establish a minimal set of functionality for the new target platform. Having too many packages increases the amount of code in the initial image, as well as the possibility of encountering problems. After the minimal set of functionality is operating properly on our new platform, packages can be added back to the new platform to extend the feature set.

Packages can be removed from the new target platform configuration using the Package Control dialog box, which is launched by selecting *Build* → *Packages* from the menu. This dialog box is shown in Figure 11.15. Additional information about adding and removing packages is covered in Chapter 11.

For our example, we start by editing `hal_powerpc_martini.cdl` and the eCos repository file `ecos.db`. An example of the modifications necessary to the file `hal_powerpc_martini.cdl` for the new *Martini* platform is shown in Code Listing 13.3.

```
1 cdl_option CYGHWR_HAL_POWERPC_BOARD_SPEED {
2 display "Development board clock speed (MHz)"
3 flavor data
4 legal_values 50
5 default_value 50
6 description "
7 MARTINI boards have various system clock speeds
8 depending on the processor fitted. Select the
9 clock speed appropriate for your board so that
10 the system can set the serial baud rate
11 correctly, amongst other things."
12 }
13 .
14 .
15 .
16 cdl_option CYGNUM_HAL_VIRTUAL_VECTOR_COMM_CHANNELS {
17 display "Number of communication channels on the board"
18 flavor data
19 calculated 2
20 }
```

**Code Listing 13.3** Example CDL script file changes in the file `hal_powerpc_martini.cdl` for the new *Martini* platform.

In Code Listing 13.3, we see an example of two of the changes to the CDL script file to match the hardware for the *Martini* platform. The first change is under the CDL command `cdl_option CYGHWR_HAL_POWERPC_BOARD_SPEED` shown on line 1. In our example, the MBX board offers the ability to run the processor clock at two different speeds, 40 and 50 MHz. On the *Martini* hardware, we only have a single processor speed 50 MHz; therefore, we change line 4, `legal_values`, to only accept 50 for the processor speed.

The other example of a change is for the CDL command `cdl_option CYGNUM_HAL_VIRTUAL_VECTOR_COMM_CHANNELS`. The MBX board only has one serial channel to use for communications; however, the new *Martini* hardware offers two. Therefore, on line 19, `calculated`, we change the value from 1 to 2.

Other CDL commands that need to be verified are the options `CYGNUM_HAL_RTC_NUMERATOR`, `CYGNUM_HAL_RTC_DENOMINATOR`, and `CYGNUM_HAL_RTC_PERIOD`, which are located under the component `CYGNUM_HAL_RTC_CONSTANTS`. These options set the real-time clock constant according to the hardware clock settings.

Options might also need to be removed completely from the CDL script file since they do not apply to the new hardware platform. An example of this is the option

CYGPKG\_HAL\_POWERPC\_MBX\_TESTS, since there are no tests we want to build for the *Martini* platform.

Now we need to verify that the proper packages are loaded for our new *Martini* platform. This is accomplished by editing the `martini` target we added to the `ecos.db` file in step 1. The reason we did not make these modifications in step 1 was so we could verify the build procedure and run the RedBoot image on the evaluation board.

We want to remove as many of the packages as possible to establish a basic level of functionality. After this basic functionality is operating properly on our new hardware platform, we can add back the packages we need to enhance the functionality of our platform. Code Listing 13.4 shows the `packages` command within the `martini` target.

```

1 packages { CYGPKG_HAL_POWERPC
2 CYGPKG_HAL_POWERPC_MPC8xx
3 CYGPKG_HAL_POWERPC_MARTINI
4 CYGPKG_HAL_QUICC
5 CYGPKG_IO_SERIAL_POWERPC_QUICC_SMC
6 CYGPKG_DEVS_ETH_POWERPC_QUICC
7 CYGPKG_DEVS_FLASH_MBX
8 CYGPKG_DEVS_FLASH_AMD_AM29XXXXX
9 }
```

**Code Listing 13.4** Example package changes to the file `ecos.db` for the new *Martini* platform.

In Code Listing 13.4, we see all the packages that will be loaded when the *Martini* board target is selected from the Templates dialog box. Some of these packages might need to be removed all together because the hardware functionality might not be included on the new hardware platform. In other cases, we might need to change a package that is loaded.

For example, if the new *Martini* platform used the PowerPC 60x variant instead of the 8xx variant, we would need to change line 2 from `CYGPKG_HAL_POWERPC_MPC8xx` to `CYGPKG_HAL_POWERPC_PPC60x`. This ensures that the PPC60x variant package is loaded for the *Martini* template.

We leave the packages `CYGPKG_HAL_QUICC` and `CYGPKG_IO_SERIAL_POWERPC_QUICC_SMC` on lines 4 and 5 because these provide the serial port communication code for our new platform. In the next step, we verify that the code configures and operates the serial port as required by our new hardware.

Since we are initially using serial communication on our new hardware platform, we want to remove line 6 so the `CYGPKG_DEVS_ETH_POWERPC_QUICC` package is not loaded. After a baseline of functionality is established on our new platform, we can add Ethernet support back.

If additional hardware is present on the new platform that is not present on the baseline HAL platform, a new package might need to be added. For example, if a different flash device is used on the *Martini* platform, we would remove line 8 `CYGPKG_DEVS_FLASH_AMD_AM29XXXXX` and substitute the flash device present on the new hardware platform.

---

**NOTE** In some platform porting cases it might be necessary to write your own device driver package if there is not support for your hardware device in the eCos repository. If you need to write your own device driver package, you can model the new package after one of the existing driver packages. The steps to do this are the same as described in this platform porting procedure.

#### STEP 4 Adjust the Memory Layout for the New Platform

The memory layout files are located in the `include\pkgconf` subdirectory in our new *Martini* package. We need to edit the `.h` and `.ldi` files to match the memory on the new hardware platform. Editing the `.mlt` file is not necessary. The `.mlt` files are used by the Configuration Tool to store the graphical information for the memory layout. There are two sets of memory layout files, one for RAM startup and one for ROM startup.

Initially we can hand edit these files. After the code is up and running on our new platform, it is best to use the Configuration Tool Memory Layout window, as described in Chapter 11 for changes to the memory structure. The memory layout editor is currently only present in version 1.3.net of the Configuration Tool. Version 2 of the Configuration Tool does not have the capability to edit memory configurations.

It is helpful to have the GNU Linker documentation present when editing these files. The documentation describes all of the linker script commands that are defined in the memory layout files. The online site for the GNU Linker (`ld`) is:

[www.gnu.org/manual/manual.html](http://www.gnu.org/manual/manual.html)

For our example platform port, the two files we need to edit are `mlt_powerpc_martini_rom.h` and `mlt_powerpc_martini_rom.ldi`. Code Listing 13.5 shows part of the ROM memory layout file `mlt_powerpc_martini_rom.h` for the new *Martini* platform.

```
1 #define CYGMEM_REGION_ram (0)
2 #define CYGMEM_REGION_ram_SIZE (0x400000)
3 #define CYGMEM_REGION_ram_ATTR (CYGMEM_REGION_ATTR_R |
 CYGMEM_REGION_ATTR_W)
4 #define CYGMEM_REGION_rom (0xfe000000)
5 #define CYGMEM_REGION_rom_SIZE (0x800000)
6 #define CYGMEM_REGION_rom_ATTR (CYGMEM_REGION_ATTR_R)
```

**Code Listing 13.5** Example memory layout file, `mlt_powerpc_martini_rom.h`, for the new *Martini* platform.

As we see in Code Listing 13.5, the RAM and ROM memory regions are defined. We need to adjust the size and start addresses of the RAM and ROM regions to match our new hardware platform. Line 1 contains the start address, set to 0, for the RAM memory region. The size of the RAM region is defined on line 2, which is set to `0x0040_0000` (4 Mbytes). Line 3 defines the properties of the RAM region as read and write. The ROM start address is defined on line 4 at

0xFE00\_0000. The size of the ROM memory, set on line 5, is 0x0080\_0000 (8 Mbytes). Finally, the attributes for the ROM region are set on line 6 to read-only.

We also need to modify the `.ldi` linker script file to match the new hardware platform. In Code Listing 13.6, we see part of the `mlt_powerpc_martini_rom.ldi` file.

```
1 MEMORY
2 {
3 ram : ORIGIN = 0, LENGTH = 0x400000
4 rom : ORIGIN = 0xfe000000, LENGTH = 0x800000
5 }
6
7 SECTIONS
8 {
9 SECTIONS_BEGIN
10 SECTION_vectors (rom, 0xfe000000, LMA_EQ_VMA)
11 SECTION_text (rom, ALIGN (0x4), LMA_EQ_VMA)
12 SECTION_fini (rom, ALIGN (0x4), LMA_EQ_VMA)
13 SECTION_rodata1 (rom, ALIGN (0x8), LMA_EQ_VMA)
14 SECTION_rodata (rom, ALIGN (0x8), LMA_EQ_VMA)
15 SECTION_fixup (rom, ALIGN (0x4), LMA_EQ_VMA)
16 SECTION_gcc_except_table (rom, ALIGN (0x1), LMA_EQ_VMA)
17 .
18 .
19 .
20 SECTION_bss (ram, ALIGN (0x10), LMA_EQ_VMA)
21 CYG_LABEL_DEFN(__heap1) = ALIGN (0x8);
22 SECTIONS_END
23 }
```

**Code Listing 13.6** Example linker script file, `mlt_powerpc_martini_rom.ldi`, for the new *Martini* platform.

The `MEMORY` linker command is on lines 1 through 5. This command describes the location and size of the regions of memory for the hardware platform. As we can see on lines 3 and 4, the `ram` and `rom` regions are defined to match the size and start addresses in the `mlt_powerpc_martini_rom.h` file.

A portion of the `SECTIONS` command is defined on lines 7 through 23, which tells the linker how to map input section into output sections and where the output sections are located in memory. We want to verify that all memory sections are located in the appropriate place for the new hardware platform. The macros, such as `SECTION_vectors`, are defined in the PowerPC linker script file `powerpc.ld` located under the `hal\powerpc\arch\current\src` subdirectory. All platforms have an architecture linker script file. Additional information about the linker script files can be found in Chapter 11.

### STEP 5 Modify the Code for the New Platform

The next step is to modify the HAL initialization code for the new hardware platform. The main platform initialization code can be found in the assembly (.S) file located under the `src` subdirectory. In our example, this is the file `martini.S`.<sup>1</sup> The routine `hal_hardware_init` contains the platform-specific code. The modifications necessary are dependent on the platform used as the baseline and the additional functionality needed by the new platform hardware.

We also need to verify the initialization code in the other files such as `hal_aux.c` and `hal_diag.c`, also located in the `src` subdirectory. Other HAL packages might have additional source files that need to be verified. Additional information about the HAL startup process can be found in Chapter 2.

Some general areas of code to modify or comment out during this phase of the porting process are common across most platforms. These general modifications include configuring the processor registers such as the real-time clock, chip select, and interrupt control, which is accomplished in the routine `hal_hardware_init`.

Another modification is to disable instruction and data caches, as well as the MMU if possible, since speed is not a concern at this point. In our example, we can prevent the caches from being enabled by undefining the macro `CYGPRI_ENABLE_CACHES` found in the file `plf_cache.h` under the `include` subdirectory.

We also need to make sure that the serial port code initializes the port appropriately for our new hardware platform. Initially it is a good idea to use a polling mode driver, rather than an interrupt driven driver. By using RedBoot, which only uses polling mode for serial communications, we have the proper communication functionality we need at this point in the porting process.

In addition, the serial communication channel needs to be hooked into the communication interface table and virtual vector table properly. Detailed information about the communication interface table and virtual vector table can be found in Chapter 4, *Virtual Vectors*. The platform used as a baseline might already hook the necessary serial communication functions into the tables, in which case we need to verify the serial port is initialized properly.

In our example, the new *Martini* platform has two serial ports whereas the MBX platform only has one. We want to initially get one serial port up and running and then add the second serial port code, which we can model after the functioning port. As we extend the functionality of our new platform by adding packages we need to verify that the code operates as intended for our hardware.

### STEP 6 Build the RedBoot Image for the New Platform

Now we are ready to build the RedBoot image for our new *Martini* platform. The new platform is built by selecting *Build* → *Library* from the menu in the Configuration Tool. The method and tools needed to load the image onto the new hardware platform are unique to the platform.

1. This is just another reminder about the file extension for the assembly file. In this case, it is important that the extension of the assembly file is an uppercase “S” rather than a lowercase “s”. The uppercase “S” allows the GNU C compiler driver to perform preprocessing on the assembly file.



It might be necessary to use the GNU Binary utility `objcopy` to convert the file to a format that is required by the device programming tools. Additional information about the `objcopy` utility can be found under `binutils` online at:

[www.gnu.org/manual/manual.html](http://www.gnu.org/manual/manual.html)

After the RedBoot image is installed onto the new target platform, the board is reset to boot the new software. If RedBoot initializes and runs successfully, the initialization message is output on the serial port, similar to the message shown in Code Listing 9.1 in Chapter 9. If this message is not received, it is necessary to verify the changes made for the new target platform.

At this point, your embedded debugging skills come into play. Toggling an I/O port pin (possibly connected to an LED) to track the progress through the code is great for low-level debugging through assembly files.

### **STEP 7 Additional Functionality**

Once the RedBoot image is up and running on the new hardware, additional functionality can be added to the configuration. Some of these additional features might include adding a reset, which can be implemented in software or using a hardware watchdog.

Testing to make sure the single-step support, which should already be present in the HAL, is operating properly can be useful when using GDB to step through code. Enabling the cache on the processor, if present, is another feature to add that can enhance program speed.

## **13.2.2 Porting Hints**

Here are some hints that can come in handy when performing your own eCos port:

- Start with the smallest amount of code possible to get a minimal RedBoot image running on your new platform. Having too many packages increases the amount of code in the initial image, as well as the possibility of encountering problems. Functionality can be added after the hardware has some basic features up and running.
- Stay up to date with the eCos source code. Tracking the changes made to the platform you use as your baseline for the port is crucial because bug fixes might be incorporated that you need to take advantage of. The `ChangeLog` files are a great source for understanding what changes are occurring with a particular platform.
- Various hardware tools, such as an ICE or BDM, can speed up the porting process by giving you better visibility into the processor's operation.
- As mentioned previously, when low-level assembly debugging is necessary, toggling an I/O port pin to track the progress through the software can be very helpful.
- A trace buffer, or trace messages output to a serial port, is useful in tracking a variable's value or other information during execution of the software.
- Assertions can be used to expose bugs or missing features in the software. Assertions are disabled by default in eCos configurations.
- Using the test applications contained in the eCos framework can divulge problems with your target platform.

### 13.3 Summary

In this chapter, we went through the procedure for porting eCos to a new platform. We started with getting eCos noticed by the Configuration Tool so that we could build our new hardware platform image. We then looked at some of the modifications that might be necessary to get eCos running on new hardware.

Although the exact steps necessary for bringing up eCos on a new hardware platform are unique to that particular hardware, this porting example gives us a basic understanding of the procedure. Finally, we finished the chapter with some hints that can be used to aid in porting eCos.

### Reference

Ganssle, Jack. "ICE Technology Unplugged." *Embedded Systems Programming* (October 1999): 103.

# Supported Processors and Evaluation Platforms

**T**he tables in this appendix list the processors and evaluation boards supported by eCos. Also listed is the manufacturer of the evaluation board and the features supported by the board. Additional information about supported processors and evaluation boards can be found online at:

<http://sources.redhat.com/ecos/hardware.html>

**Table A.1**    ARM Architecture

| Processor | Evaluation Board | Manufacturer            | Supported Features                                                          |
|-----------|------------------|-------------------------|-----------------------------------------------------------------------------|
| ARM710T   | CMA222           | Cogent Computer Systems | Serial driver                                                               |
| ARM9      | Excalibur        | Altera                  | Flash driver                                                                |
| ARM9      | AAED2000         | Agilent                 | Flash driver, Ethernet driver, Keyboard driver, Touchscreen driver, RedBoot |
| ARM940T   | EPI Dev9         | Embedded Performance    | Serial driver, Flash driver, RedBoot                                        |

**Table A.1** ARM Architecture (Continued)

| <b>Processor</b>                          | <b>Evaluation Board</b>                    | <b>Manufacturer</b>     | <b>Supported Features</b>                                                          |
|-------------------------------------------|--------------------------------------------|-------------------------|------------------------------------------------------------------------------------|
| ARM7TDMI                                  | CMA230                                     | Cogent Computer Systems | Serial driver                                                                      |
| ARM7TDMI                                  | EPI Dev7                                   | Embedded Performance    | Serial driver, Flash driver, RedBoot                                               |
| ARM7TDMI                                  | ARM Integrator                             | ARM                     | Serial driver, Ethernet driver, Flash driver, PCI driver                           |
| Atmel AT91x40<br>(contains ARM7TDMI core) | AT91EB40 Evaluation Board                  | Atmel                   | Serial driver, Flash driver, RedBoot                                               |
| Cirrus Logic CL-PS7111                    | CL-PS7111 Development Board                | Cirrus Logic            | Serial driver, Ethernet driver, Flash driver                                       |
| Cirrus Logic Maverick EP7209              | EDB7209                                    | Cirrus Logic.           | Serial driver, Ethernet driver, Flash driver                                       |
| Cirrus Logic Maverick EP7211              | EDB7211                                    | Cirrus Logic            | Serial driver, Ethernet driver, Flash driver, RedBoot                              |
| Cirrus Logic Maverick EP7212              | EDB7212                                    | Cirrus Logic            | Serial driver, Ethernet driver, Flash driver, RedBoot                              |
| Intel StrongARM SA-110                    | EBSA-285                                   | Intel                   | Serial driver, Ethernet driver, Flash driver, PCI driver, Watchdog driver, RedBoot |
| Intel StrongARM SA-1100                   | Intel SA-1100 Evaluation Platform (Brutus) | Intel                   | Serial driver                                                                      |

**Table A.1** ARM Architecture (Continued)

| <b>Processor</b>              | <b>Evaluation Board</b>                                    | <b>Manufacturer</b>     | <b>Supported Features</b>                                                                                                   |
|-------------------------------|------------------------------------------------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Intel StrongARM SA-1100       | Intel SA-1100 Multimedia Board                             | Intel                   | Serial driver, Flash driver, RedBoot                                                                                        |
| Intel StrongARM SA-1110       | Intel SA-1110 Microprocessor Evaluation Platform (Assabet) | Intel                   | Serial driver, Ethernet driver, Flash driver, USB driver, PCMCIA/CF driver, Watchdog driver, RedBoot                        |
| Intel StrongARM SA-1110       | Compaq iPAQ PocketPC                                       | Hewlett-Packard         | Serial driver, Ethernet driver, Flash driver, PCMCIA/CF driver, keypad driver, touchscreen driver, watchdog driver, RedBoot |
| Intel StrongARM SA-1110       | Bright Star Engineering commEngine                         | Bright Star Engineering | Serial driver, Ethernet driver, Flash driver, PCI driver, Watchdog driver, RedBoot                                          |
| Intel StrongARM SA-1110       | CerfBoard                                                  | Intrinsyc               | Serial driver, Ethernet driver, RedBoot                                                                                     |
| Intel StrongARM SA-1110       | CerfCube                                                   | Intrinsyc               | Serial driver, Ethernet driver, RedBoot                                                                                     |
| Intel XScale 80321            | IQ80321                                                    | Intel                   | Serial driver, Ethernet driver, Flash driver, PCI driver, RedBoot                                                           |
| Samsung KS32C5000 or S3C4510A | SNDS100                                                    | Samsung                 | Serial driver, Ethernet driver, RedBoot                                                                                     |

**Table A.1** ARM Architecture *(Continued)*

| <b>Processor</b>                            | <b>Evaluation Board</b> | <b>Manufacturer</b> | <b>Supported Features</b>      |
|---------------------------------------------|-------------------------|---------------------|--------------------------------|
| Samsung KS32C50100 (contains ARM7TDMI core) | Evaluator-7T            | ARM                 | RedBoot                        |
| Sharp LH77790                               | ARM AEB-1               | ARM                 | Serial driver, Watchdog driver |

**Table A.2** Samsung CalmRISC Architecture

| <b>Processor</b> | <b>Evaluation Board</b> | <b>Manufacturer</b> | <b>Supported Features</b> |
|------------------|-------------------------|---------------------|---------------------------|
| CalmRISC16       | Calm16 Core             | Samsung             | RedBoot                   |
| CalmRISC32       | Calm32 Core             | Samsung             | RedBoot                   |

**Table A.3** Hitachi H8/300 Architecture

| <b>Processor</b> | <b>Evaluation Board</b>  | <b>Manufacturer</b> | <b>Supported Features</b>               |
|------------------|--------------------------|---------------------|-----------------------------------------|
| Hitachi H8/300   | H8/3068 Evaluation Board | Akizuki             | Serial driver, Ethernet driver, RedBoot |

**Table A.4** Fujitsu FR-V Architecture

| <b>Processor</b> | <b>Evaluation Board</b> | <b>Manufacturer</b> | <b>Supported Features</b>                |
|------------------|-------------------------|---------------------|------------------------------------------|
| FR-V             | FR-V400                 | Fujitsu             | Flash drivers, Ethernet drivers, RedBoot |

**Table A.5** Intel x86 Architecture

| Processor | Evaluation Board | Manufacturer | Supported Features                                  |
|-----------|------------------|--------------|-----------------------------------------------------|
| x86       | PC Motherboard   | Multiple     | Serial driver, Ethernet driver, PCI driver, RedBoot |

**Table A.6** Matsushita AM3x Architecture

| Processor        | Evaluation Board       | Manufacturer                             | Supported Features                                    |
|------------------|------------------------|------------------------------------------|-------------------------------------------------------|
| Panasonic AM31   | stdeval1               | Syoichi Yamamoto<br>Kyoto Micro Computer | Serial driver,<br>Watchdog driver                     |
| Panasonic AM33   | STB Reference Platform | Syoichi Yamamoto<br>Kyoto Micro Computer | Serial driver,<br>Watchdog driver                     |
| Panasonic AM33-2 | ASB2305                | Matsushita                               | Serial driver, Ethernet driver, Flash driver, RedBoot |

**Table A.7** MIPS Architecture

| Processor | Evaluation Board | Manufacturer      | Supported Features                                                |
|-----------|------------------|-------------------|-------------------------------------------------------------------|
| MIPS 4Kc  | Atlas            | MIPS Technologies | Serial driver, Ethernet driver, Flash driver, PCI driver, RedBoot |
| MIPS 4Kp  | Atlas            | MIPS Technologies | Serial driver, Ethernet driver, Flash driver, RedBoot             |
| MIPS 4Km  | Atlas            | MIPS Technologies | Serial driver, Ethernet driver, Flash driver, RedBoot             |

**Table A.7** MIPS Architecture (Continued)

| <b>Processor</b>   | <b>Evaluation Board</b>  | <b>Manufacturer</b>         | <b>Supported Features</b>                                         |
|--------------------|--------------------------|-----------------------------|-------------------------------------------------------------------|
| MIPS 4Kc           | Malta                    | MIPS Technologies           | Serial driver, Flash driver, Ethernet driver RedBoot              |
| MIPS 5Kc           | Malta                    | MIPS Technologies           | Serial driver, Flash driver, Ethernet driver RedBoot              |
| NEC VR4300         | DDB-VRC4373              | NEC Electronics             | Serial driver, PCI driver                                         |
| NEC VRC4375        | Blue Nile                | NEC Electronics             | Serial driver, Flash driver, Ethernet driver, RedBoot             |
| PMC-Sierra RM7000A | Ocelot                   | Momentum Computer           | Serial driver, Ethernet driver, Flash driver, PCI driver, RedBoot |
| Toshiba TMPR3904   | JMR-TX3904 (Japanese)    | Toshiba Information Systems | Serial driver                                                     |
| Toshiba TMPR4955F  | TMPR4955 Reference Board | Toshiba Information Systems | Serial driver, Real-time clock driver                             |

**Table A.8** NEC V8xx Architecture

| <b>Processor</b> | <b>Evaluation Board</b>       | <b>Manufacturer</b> | <b>Supported Features</b> |
|------------------|-------------------------------|---------------------|---------------------------|
| NEC V850/SA1     | Cosmo CEB-V850/SA1 (Japanese) | Cosmo (Japan)       | Serial driver             |
| NEC V850/SB1     | Cosmo CEB-V850/SB1 (Japanese) | Cosmo (Japan)       | Serial driver             |



**Table A.9** Table A.9 PowerPC Architecture

| <b>Processor</b>                  | <b>Evaluation Board</b>      | <b>Manufacturer</b>                   | <b>Supported Features</b>                                    |
|-----------------------------------|------------------------------|---------------------------------------|--------------------------------------------------------------|
| Motorola<br>MCF527C23<br>ColdFire | M5272C23<br>Evaluation Board | Motorola<br>Semiconductor<br>Products | Serial driver, Ethernet<br>driver, Flash driver,<br>RedBoot  |
| Motorola MPC555                   | CMx-555                      | Axiom<br>Manufacturing                | Serial driver, Flash<br>driver, Wallclock<br>driver, RedBoot |
| Motorola MPC823                   | CMA287-23                    | Cogent Computer<br>Systems            | Serial driver                                                |
| Motorola MPC850                   | CMA287-50                    | Cogent Computer<br>Systems            | Serial driver                                                |
| Motorola MPC860                   | CMA286-60                    | Cogent Computer<br>Systems            | Serial driver                                                |
| Motorola MPC860                   | MPC8xxFADS                   | Motorola<br>Semiconductor<br>Products |                                                              |
| Motorola MPC860                   | MBX860                       | Motorola Computer<br>Group            | Serial driver, Ethernet<br>driver, Flash driver,<br>RedBoot  |
| Motorola MPC860T                  | Viper                        | Analogue & Micro                      | Serial driver, Ethernet<br>driver, Flash driver,<br>RedBoot  |

**Table A.10** SPARC Architecture

| <b>Processor</b>                          | <b>Evaluation Board</b> | <b>Manufacturer</b>         | <b>Supported Features</b> |
|-------------------------------------------|-------------------------|-----------------------------|---------------------------|
| Fujitsu MB86831,<br>MB86832 or<br>MB86833 | Fujitsu<br>MB86800-MA01 | Fujitsu<br>Microelectronics | Serial driver             |

**Table A.11** SuperH Architecture

| <b>Processor</b> | <b>Evaluation Board</b>     | <b>Manufacturer</b>      | <b>Supported Features</b>                                                                                     |
|------------------|-----------------------------|--------------------------|---------------------------------------------------------------------------------------------------------------|
| Hitachi SH7708   | Hitachi EDK/<br>SH7708      | Hitachi<br>Semiconductor | Serial driver, Real-time<br>clock driver, Watchdog<br>driver                                                  |
| Hitachi 7729     | Hitachi<br>HS7729PCI        | Hitachi<br>Semiconductor | Serial driver, Flash driver,<br>Ethernet driver, PCI<br>driver, Watchdog driver,<br>Wallclock driver, RedBoot |
| Hitachi 7709     | Solution Engine             | Hitachi<br>Semiconductor | Serial driver, Flash driver,<br>Ethernet driver, Watchdog<br>driver, RedBoot                                  |
| Hitachi SH7708   | CqREEK SH7708<br>(Japanese) | CQ Publishing<br>(Japan) | Serial driver                                                                                                 |
| Hitachi SH7750   | Sega Dreamcast              | Sega                     | Serial driver, PCI driver,<br>RedBoot                                                                         |
| Hitachi SH7750   | CqREEK SH7750<br>(Japanese) | CQ Publishing<br>(Japan) | Serial driver                                                                                                 |
| Hitachi SH7751   | Solution Engine             | Hitachi<br>Semiconductor | Serial driver, Flash driver,<br>Ethernet driver, Watchdog<br>driver, RedBoot                                  |

## **eCos License**

**e**Cos version 2 is released under a GNU Public License (GPL) compatible license. Previous versions of eCos were covered by the Red Hat eCos Public License (RHEPL); however, with the latest release, the eCos license was changed.

The first part of this appendix describes the eCos license, which is found in the source files of the eCos source code repository. The GPL version 2 follows the eCos license. The GPL can be found online at:

[www.gnu.org/licenses/gpl.html](http://www.gnu.org/licenses/gpl.html)

The eCos license is the standard GPL version 2 but with a special exception to make it appropriate for embedded systems. In summary, this license means that when distributing products and binaries containing eCos, you must include or make available the source code to eCos that you used, for example on your Web site. However, there is nothing in the eCos license that requires you to release source code for your own software—only the source code of eCos, or derived from eCos. The details of the license are covered below.

Additional information about the eCos license can be found on the eCos discussion list. One specific message that interprets some of the details about the new eCos license can be found online at:

<http://sources.redhat.com/ml/ecos-discuss/2002-05/msg00191.html>

### **B.1 eCos License**

This file is part of eCos, the Embedded Configurable Operating System.  
Copyright (C) 1998, 1999, 2000, 2001, 2002 Red Hat, Inc.

eCos is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 or (at your option) any later version.

eCos is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with eCos; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

As a special exception, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other works to produce a work based on this file, this file does not by itself cause the resulting work to be covered by the GNU General Public License. However, the source code for this file must still be made available in accordance with section (3) of the GNU General Public License.

This exception does not invalidate any other reasons why a work based on this file might be covered by the GNU General Public License.

Alternative licenses for eCos may be arranged by contacting Red Hat, Inc. at <http://sources.redhat.com/ecos/ecos-license/>

## **B.2 GNU GENERAL PUBLIC LICENSE**

### **B.2.1 Version 2, June 1991**

Copyright (C) 1989, 1991 Free Software Foundation,  
Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### **B.2.2 Preamble**

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

#### GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution, and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.



Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### END OF TERMS AND CONDITIONS

### B.2.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
Also add information on how to contact you by electronic and paper mail.
```

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY;
for details type 'show w'.
```

```
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other

than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program  
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 21 April 2002  
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# Cygwin Tools Upgrade Procedure

**T**he Cygwin tools upgrade procedure allows you to take advantage of bug fixes and enhancements to the tools installed from the CD-ROM in Chapter 10, *Cygwin Tools Upgrade Procedure*.

For this Cygwin tools upgrade example, we are running through the process of updating the `cygwin` package from the Internet. Other Cygwin packages might be newer than the ones initially installed because updates and bug fixes are constantly occurring for each package.

---

**NOTE** A good idea prior to upgrading any package is to search the eCos discussion mailing list to see if anyone has reported a problem with a particular package. You should also perform the upgrade in a manner so that you can revert to your previous working tools if the upgrade has problems. Posting messages on the eCos discussion mailing list, or the relevant Cygwin mailing list, is another way to obtain help.

Before beginning the Internet upgrade, you should check with your network administrator to determine if there are any special configuration options you need to be aware of, such as proxy settings. The setup program prompts you for the type of Internet connection you have on your system.

## STEP 1

The first step in the upgrade process is to run the `setup.exe` program. This is the same program we used in the initial installation of the Cygwin tools. The same dialog box shown in Figure 10.1 is displayed, provided the same `setup.exe` is used. To proceed, click the Next button.

**STEP 2**

For the upgrade procedure, we use the *Install from Internet* option, shown in Figure 10.2. It is possible to download the latest packages and then install from the downloaded location, as we did in the initial installation. However, using the *Install from Internet* option allows us to perform the download and install tasks in consecutive steps. Then, click the Next button.

**STEP 3**

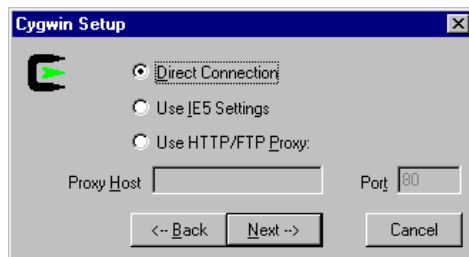
Next, we set the *Local Package Directory* to `D:\cygwin`, as shown in Figure 10.3. Click the Next button.

**STEP 4**

The next step is the selection of the location, for the *Select Install Root Directory* option, where the tools should be installed. This is shown in Figure 10.4. We use `D:\cygwin` along with the options *DOS* for the *Default Text File Type* option, and *Just Me* for the *Install For* option. After making these option selections, click the Next button.

**STEP 5**

We now need to choose the type of Internet connection for downloading the Cygwin package. The dialog box displaying the download options is shown in Figure C.1. The option selected depends on your specific connection to the Internet. You might need to check with your network administrator to ensure the proper option is selected. Select *Direct Connection*, and click Next to proceed.



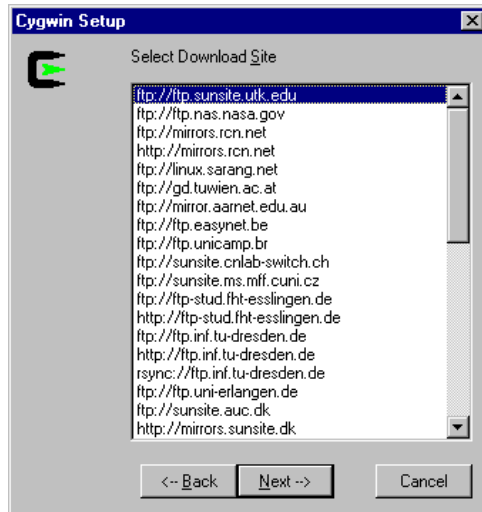
**Figure C.1** Cygwin Internet Connection Options dialog box.

**STEP 6**

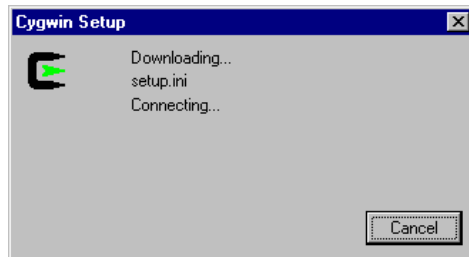
Next, we select the mirror site to use for our file downloads for the *Select Download Site* option. This dialog box is shown in Figure C.2. The selection for this option depends on your location. It is typically best to select a download site close to your facility. After selecting your download site, click the Next button.

**STEP 7**

The dialog box shown in Figure C.3 is displayed after the download site is selected. This dialog box remains open until the connection to the download site is complete.



**Figure C.2** Cygwin Download Site dialog box.




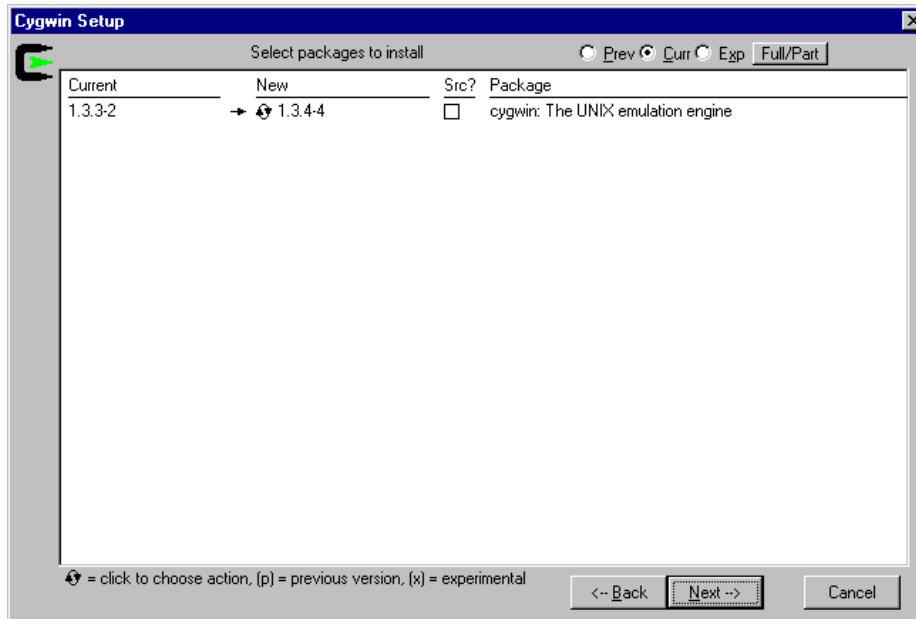
**Figure C.3** Cygwin download connection dialog box.

After successfully connecting to the download site, the dialog box shown in Figure C.4 appears. As we can see from this dialog box, the current version of the `cygwin` package we have installed is 1.3.3-2, as shown in the *Current* column. The new version available on the download site is 1.3.4-4, as shown in the *New* column.

---

**NOTE** The package selection dialog box you see might differ from the one shown in Figure C.4, because the Cygwin tools are updated constantly. Therefore, any package that is newer than the one we installed in Chapter 10 will show up in the Package Selection dialog box.

If you want to leave a particular version of a package alone, click on the  until *Skip* is displayed in the *New* column. This prevents the setup program from overwriting the current version of the package you installed.



**Figure C.4** Cygwin Package Selection dialog box.

After clicking the Next button to continue, the packages selected are downloaded and installed. A dialog box similar to the one shown in Figure 10.6 is displayed during download and installation.

### STEP 8

When the installation completes, the dialog box shown in Figure 10.7 is displayed. Since we already have our shortcuts configured, we can leave both options unchecked and click Next. Finally, the dialog box shown in Figure 10.8 is displayed, indicating successful upgrade of the Cygwin packages. Click OK and we are ready to use the latest Cygwin tools.

### UPGRADE HINTS

Here are a few hints to overcome some common problems during the upgrade procedure. If you are having problems connecting to a download site, simply select a different mirror site. Often times, the mirror sites are being updated so a particular site might be down for some time.

If the setup program is having troubles upgrading a particular package, you might need to edit the `install.db` file located in the `etc\setup` directory under the `cygwin` root directory. Locate the package or packages you are upgrading and delete the text line for those particular packages. It is best to create a backup copy of the `install.db` file prior to doing any editing.

Check the log files for additional information regarding the upgrade. The setup program writes details about the setup procedure, which might assist you in finding the installation problem.

Finally, check the Cygwin mailing list for specific help with any setup problems. It is helpful to give the `setup.exe` program version number (in our case, the version number is 2.78.2.15) when posting to this list. A list of Cygwin mailing lists can be found online at:

<http://sources.redhat.com/cygwin/lists.html>



# Building the GNU Cross-Development Tools

**T**he procedure detailed in this appendix was used to build the i386 GNU cross-development tools contained on the CD-ROM. The commands in this section are specific to configuring and building the i386 GNU cross-development tools, although the commands for other processors are very similar.

The configure and build process for each group of GNU cross-development tools can take a large amount of time to complete. The amount of time to configure and build the cross-development tools directly depends on the speed of your PC.

---

**NOTE** The steps detailed in this build procedure assume that a new host development platform is being used. Therefore, some of the steps—for example, the mount command in step 4—might not be necessary if new development tools are being built on an existing platform.

## STEP 1

Open the bash command shell. This can be done by clicking on the Cygwin shortcut on the desktop, if created in the Cygwin native tools installation, or through the menu *Start -> Programs -> Cygnus Solutions -> Cygwin Bash Shell*.

When the shell is opened, the present working directory is `D:\cygwin\home\xxx`, where `xxx` is your username. We want to change to the root Cygwin directory, which in our case is `D:\cygwin`, by entering the following command at the bash prompt (`$`):

```
$ cd /
```

## STEP 2

Next, we need to create subdirectories for each of the three groups of cross-development tools. When using the Cygwin bash shell, the “/” refers to the root Cygwin installation directory, which in our case is `D:\cygwin`. To create the necessary directories, enter the following command:

```
$ mkdir -p /src/binutils /src/gcc /src/gdb
```

Verify that these directories are created correctly under the `D:\cygwin` root directory.

## STEP 3

---

**NOTE** The CD-ROM drive is mounted as `/cygdrive/e/` by default when Cygwin is installed. The drive letter `f` for your CD-ROM should be substituted in place of `/e/` in the following `tar` commands. If you are uncertain of the drive mountings for your system, enter the command `mount` at the bash shell prompt to get a listing of the current system mounts.

### *GNU Binary Utilities Unzip*

Now we need to unzip the source code files from the CD-ROM. First, we change to the GNU Binutils directory with the command:

```
$ cd /src/binutils
```

Then, we unzip the file with the command:

```
$ tar xjvf /cygdrive/e/gnu/source/binutils-2.11.2a.tar.bz2
```

This creates the subdirectory `binutils-2.11.2a`, under the `src/binutils` directory, which contains the GNU Binutils source files.

### *GNU C/C++ Compiler Unzip*

Next, we change to the GNU C/C++ Compiler subdirectory at the bash shell prompt with the command:

```
$ cd /src/gcc
```

Then, unzip the file with the command:

```
$ tar xjvf /cygdrive/e/gnu/source/gcc-2.95.2a.tar.bz2
```

This creates the subdirectory `gcc-2.95.2a`, under the `src/gcc` directory, containing the GCC and G++ source files.

### *GNU Debugger with Insight Unzip*

Finally, we change to the GNU Debugger with Insight subdirectory with the command:

```
$ cd /src/gdb
```

Unzip the file with the command:

```
$ tar xjvf /cygdrive/e/gnu/source/insight-5.1a.tar.bz2
```

This creates the subdirectory `insight-5.1a` under the `src/gdb` directory with the GNU Debugger with Insight source files.

---

**NOTE** It is possible to install the GNU Debugger without the Insight GUI; however, the Insight source code is a superset of the GDB source code. In addition, you always have the option of running the GNU Debugger with the command-line interface using the `-nw` option.

#### STEP 4

Before building the cross-development tools, we need to make sure that the temporary directory is mounted properly on our system. This ensures that the cross development tools build properly. The command to do this is:

```
$ mount -f -b d:/cygwin/tmp /tmp
```

#### STEP 5

Now we are ready to build the cross-development tools.

---

**NOTE** In the following configure and build steps, entering the commands exactly as shown is very important. The slightest typo in entering these commands can cause the entire GNU cross-development tool chain to fail to work.

If there are problems during the build, the bash shell history is a good place to start to see if the GNU cross-development tools were built properly. The bash command history can be displayed by entering the following command:

```
$ history
```

The bash command history can also be found in the file `.bash_history` in the `D:\cygwin\home\username` directory, where `username` is your computer user name.

We start with the GNU Binutils. First, we need to create a temporary directory for the build using the bash shell command:

```
$ mkdir -p /tmp/build/binutils
```

We should verify that the directory `D:\cygwin\tmp\build\binutils` is created. Change to the subdirectory we just created with the command:

```
$ cd /tmp/build/binutils
```

Next, we configure the system to build the GNU Binutils with the command:

---

**NOTE** In the following commands, the backslash (\) at the end of each line is used to break up the commands entered into the bash shell. After pressing Enter at the end of a line with a backslash, a new line is output in the bash shell with a greater than (>) sign prompt, allowing us to continue entering the remaining command lines.

```
$ /src/binutils/binutils-2.11.2a/configure --target=i386-elf \
--prefix=/tools \
--exec-prefix=/tools/H-i686-pc-cygwin \
-v 2>&1 | tee configure.out
```

The output messages from the configuration process are contained in the file `configure.out` located in the `D:\cygwin\tmp\build\binutils` subdirectory should you need to refer to a particular message.

After the configuration has completed, we are returned to the bash shell prompt. The configuration populates the `D:\cygwin\tmp\build\binutils` subdirectory with the necessary files to build the GNU Binutils.

Now we can build the GNU Binutils with the command:

```
$ make -w all install 2>&1 | tee make.out
```

The output from the make process is contained in the file `make.out` located in the `D:\cygwin\tmp\build\binutils` subdirectory should you need to refer to a particular message.

After this step is complete, in which case we are returned to the bash shell prompt, we have the GNU Binutils created for the Intel x86 platform. These files are located in the `D:\cygwin\tools\H-i686-pc-cygwin\bin` subdirectory. An example of one of the GNU Binutils files for the Intel x86 platform is `i386-elf-as.exe`.

## STEP 6

Before we proceed with the build process, we need to make sure that the path is configured properly and that the binary utilities are at the head of the path. To do this, we use the bash shell command:

```
$ PATH=/tools/H-i686-pc-cygwin/bin:$PATH ; export PATH
```

We also need to add the GNU Intel x86 tools directory to the Windows environment path. The path is altered by right-clicking on the My Computer icon on the desktop. This brings up a drop-down list of options. Select Properties from the drop-down list.

The System Properties dialog box is displayed. Select the Environment tab. Under the User Variables, select path. In the Value edit box, to the front of the path, add:

```
D:\cygwin\tools\H-i686-pc-cygwin\bin;
```

Then, click the Set button. Finally, click the OK button.

## STEP 7

Now we verify that the GNU Binutils were built properly and that the `PATH` is set up correctly before proceeding with the build. To do this, we enter the command:

```
$ i386-elf-as --version
```

The following message is output if everything is set up properly:

```
GNU assembler 2.11.2
Copyright 2001 Free Software Foundation, Inc.
This program is free software; you may redistribute it under
the terms of the GNU General Public License. This program has
absolutely no warranty.
This assembler was configured for a target of `i386-elf'.
```

If the message is incorrect, go back and verify that the `PATH` is configured properly. If so, you need to verify that the GNU Binutils configure and make commands were entered correctly. If they were entered incorrectly, it is best to remove the contents under the `D:\cygwin\tmp\build\binutils` directory before attempting to configure and make the GNU Binutils again.

## STEP 8

Next, we build the GNU C/C++ Compiler. Then, we create a temporary directory for the GNU C/C++ Compiler build with the command:

```
$ mkdir -p /tmp/build/gcc
```

We should verify that the directory `D:\cygwin\tmp\build\gcc` is created. Change to the directory we just created using the command:

```
$ cd /tmp/build/gcc
```

Now we can configure to build the GNU C/C++ Compiler using the command:

```
$ /src/gcc/gcc-2.95.2a/configure --target=i386-elf \
--prefix=/tools \
--exec-prefix=/tools/H-i686-pc-cygwin \
--with-gnu-as --with-gnu-ld --with-newlib \
-v 2>&1 | tee configure.out
```

The output messages from the GNU C/C++ Compiler configuration process are contained in the file `configure.out` located in the `D:\cygwin\tmp\build\gcc` subdirectory should you need to refer to a particular message.

After the configuration has completed, we are returned to the bash shell prompt. The configuration populates the `D:\cygwin\tmp\build\gcc` subdirectory with the necessary files to build the GNU C/C++ Compiler.

Now we can build the GNU C/C++ Compiler with the command:

```
$ make -w all-gcc install-gcc \
LANGUAGES="c c++" 2>&1 | tee make.out
```

The output from the GNU C/C++ Compiler make process is contained in the file `make.out` located in the `D:\cygwin\tmp\build\gcc` subdirectory should you need to refer to a particular message.

After this step is complete, in which case we are returned to the bash shell prompt, we have the GNU C/C++ Compiler created for the Intel x86 platform. These files are located in the `D:\cygwin\tools\H-i686-pc-cygwin\bin` subdirectory. An example of one of the GNU C/C++ Compiler files for the Intel x86 platform is `i386-elf-gcc.exe`.

### STEP 9

Now we verify that the GNU C/C++ Compiler was built properly. To do this, we enter the command:

```
$ i386-elf-gcc --version
```

The following message is output if everything is set up properly:

```
2.95.2
```

If the message is incorrect, you need to verify that the GNU C/C++ Compiler configure and make commands were entered correctly. If they were entered incorrectly, it is best to remove the contents under the `D:\cygwin\tmp\build\gcc` directory before attempting to configure and make the GNU C/C++ Compiler again.

### STEP 10

Finally, we build the GNU Debugger with Insight. First, we create a temporary directory for the GNU Debugger build with the command:

```
$ mkdir -p /tmp/build/gdb
```

We should verify that the directory `D:\cygwin\tmp\build\gdb` is created. Change to the directory we just created using the command

```
$ cd /tmp/build/gdb
```

Now we can configure to build the GNU Debugger with Insight using the command:

```
$ /src/gdb/insight-5.1a/configure --target=i386-elf \
--prefix=/tools \
--exec-prefix=/tools/H-i686-pc-cygwin \
-v 2>&1 | tee configure.out
```

The output messages from the configuration process are contained in the file `configure.out` located in the `D:\cygwin\tmp\build\gdb` subdirectory should you need to refer to a particular message.

After the configuration has completed, we are returned to the bash shell prompt. The configuration populates the `D:\cygwin\tmp\build\gdb` subdirectory with the necessary files to build the GNU Debugger with Insight.

Now we can build the GNU Debugger with Insight with the command:

```
$ make -w all install CC='gcc -mwin32' 2>&1 | tee make.out
```

The output from the make process is contained in the file `make.out` located in the `D:\cygwin\tmp\build\gdb` subdirectory should you need to refer to a particular message.

After this step is complete, in which case we are returned to the bash shell prompt, we have the GNU Debugger with Insight Interface created for the Intel x86 platform. These files are located in the `D:\cygwin\tools\H-i686-pc-cygwin\bin` subdirectory. An example of one of the GNU Debugger with Insight files for the Intel x86 platform is `i386-elf-gdb.exe`.

## STEP 11

Now we verify that the GNU Debugger with Insight was built properly. To do this, we enter the command:

```
$ i386-elf-gdb --version
```

The following message is output if everything is set up properly:

```
GNU gdb 5.1
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public
License, and you are welcome to change it and/or distribute
copies of it under certain conditions. Type "show copying" to
see the conditions.
There is absolutely no warranty for GDB. Type "show warranty"
for details.
This GDB was configured as "--host=i686-pc-cygwin --
target=i386-elf".
```

If the message is incorrect, you need to verify that the GNU Debugger with Insight configure and make commands were entered correctly. If they were entered incorrectly, it is best to remove the contents under the `D:\cygwin\tmp\build\gdb` directory before attempting to configure and make the GNU Debugger with Insight again.

## Other Processor Tools

As mentioned previously, the configure and make commands given for building the cross-compiler tools are specific for the Intel x86 processor, although the commands for the other supported processors are very similar. The installation procedures of the GNU cross-development tools for other processors can be found online at:

<http://sources.redhat.com/ecos/getstart.html>

The GNU cross-development tools for other processors can coexist with each other as long as the tool executable files are not the same. This might be necessary if you are developing eCos applications for multiple processor architectures. Additional information about selecting the build and user tools with the Configuration Tool can be found in Chapter 11, *The eCos Toolset*.

### **Conserving Space**

If conserving disk space is an issue, you can delete the directories used during the build procedure. The directories that can be safely removed are:

- D:\cygwin\src\binutils
- D:\cygwin\src\gcc
- D:\cygwin\src\gdb

The following directories can also be deleted; however, it might be useful to save the configure and make output files generated during the build:

- D:\cygwin\tmp\build\binutils
- D:\cygwin\tmp\build\gcc
- D:\cygwin\tmp\build\gdb



---

## A B O U T T H E A U T H O R

**ANTHONY J. MASSA** earned a dual B.S./B.A. degree in electrical engineering from the University of San Diego. He has spent most of his career developing embedded software, device drivers, and applications on a wide range of 8-, 16-, and 32-bit processors and numerous embedded RTOS platforms. Anthony also has experience with hardware design, and with writing drivers and application software on the different Microsoft Windows operating system platforms.

Anthony has developed his skills by working on a diverse array of successful products including satellite PC receiver cards, set-top boxes, Internet-enabled wireless modems, networking broadcast equipment, and cable modems. He has written extensively on eCos in leading publications including the magazines *Doctor Dobb's Journal*, *Embedded Systems Programming*, *Software Development*, and *EDN*.

When not working, Anthony enjoys spending time with his family at home or on vacation (where he can often be found having a good time at the Buena Vista in San Francisco), getting out in the backcountry with his grandfather, and playing a little golf. Now that this project is concluded, he is looking for his next writing adventure—perhaps a novel!

---

# I N D E X

## A

alarms, 125, 130–33  
    example, 127, 132  
    handler, 130  
    kernel API, 131–32  
architecture. *See* Hardware Abstraction  
    Layer (HAL); submodules  
asserts and tracing, 74, 134–38, 297  
    configuration, 137–38  
    example, 136–37  
    trace macros, 134

## B

Background Debug Mode (BDM), 317, 334  
binutils. *See* GNU Binary Utilities (binutils)  
bitmap scheduler. *See* kernel; scheduler  
Bootstrap Protocol (BOOTP), 172, 175, 194  
bug tracking (Bugzilla), 8  
build process, 282–85  
building blocks, 8, 10

## C

CDL. *See* Component Definition  
    Language (CDL)  
ChangeLog file, 241, 266, 334

clocks, 129–30  
    configuration, 122  
    example, 133  
    HAL macros, 122  
    kernel API, 129  
    Real-Time Clock (RTC), 122–24  
    tick calculation, 123  
command-line configuration tool, 248, 277  
    building, 277  
communication (COMMS) channels, 67–69  
Communication Interface Table (CIT), 67–71.  
    *See also* communication  
    (COMMS) channels  
compatibility layers, 12, 150–52  
     $\mu$ ITRON, 16, 152  
    POSIX, 15, 150  
        configuration, 151  
        EL/IX, 15, 151  
        file I/O, 156–57  
compiler flags, 284–85, 301, 305  
Component Definition Language (CDL),  
    13, 240, 243–47, 256  
commands, 244  
script files, 13, 239, 243–47, 256  
    example, 247  
    graphical representation, 266–70

- component framework, 3, 8–10, 13, 239, 240
  - component repository, 10–13, 244, 264, 274
    - directory structure, 11–13
    - online, 12
  - components, 14
  - Concurrent Versions System (CVS), 12, 229
  - condition variables, 105–9
    - configuration, 105
    - example, 107–9
    - kernel API, 106–7
  - configuration method, 4–5
  - configuration options, 10, 13–14
    - nesting, 13
    - suboptions, 10, 13, 26
  - Configuration Tool, 10, 248–75
    - build options, 284–85
    - building, 277
    - CDL script files, 266–70, 271
    - Configuration window, 250, 254–55
    - conflicts and resolutions, 272–73
    - Conflicts window, 255, 273
    - eCos Configuration file (.ecc), 9, 251–53
    - eCos Minimal Configuration file (.ecm), 188, 253
      - example, 254
    - file generation, 282–84
    - importing and exporting, 253–54
    - installation, 224–28
    - Memory layout window 256–57. *See also* Memory Layout Tool (MLT)
    - menu bar, 250
    - Output window, 256
    - pop-up menu, 255
    - Properties window, 250, 255, 256, 267, 270
    - saving configuration, 251–53. *See also* eCos Configuration file (.ecm)
      - example, 252–53
    - screen layout, 248–51
    - searching, 251
    - setting build and user tools, 227, 251, 272, 327
    - Short description window, 256
    - status bar, 251
    - tests, 310–12
    - title bar, 250
    - tool bar, 251
    - versions, 248, 257
    - working directory, 252, 282–84, 292
  - conflicts, 14
  - counters, 125–29
    - configuration, 125
    - example, 127–29
    - kernel API, 126–27
  - CVS. *See* Concurrent Versions System (CVS)
  - CygMon, 12, 15, 153, 154, 185
  - Cygnus Solutions, 1–2
  - Cygwin, 209
    - cygwin.dll file, 218
    - directory structure, 217–18
    - installation, 210–17
      - log files, 218–19
    - upgrade, 219–20, 355–59
- D**
- database. *See* repository database
  - database file. *See* ecos.db (database) file
  - Deferred Service Routine (DSR), 40–41, 43, 45–47, 146–47
    - configuration, 43–44
    - example, 48–50
    - explicit posting, 50
  - development hardware setup, 286–87
  - device drivers, 12, 139, 141, 146–47, 274
  - distribution files (.epk). *See* packages; distribution files (.epk)
  - Domain Name System (DNS). *See* networking; Domain Name System (DNS)
  - Dynamic Host Configuration Protocol (DHCP), 171, 172, 175–76
  - DSR. *See* Deferred Service Routine (DSR)
  - dynamic loader, 183
- E**
- eCos Configuration file (.ecc). *See* Configuration Tool; eCos Configuration file (.ecc)

eCos development kit, 209  
  directory structure, 229  
  installation, 223–229  
eCos Minimal Configuration file (.ecm). *See*  
  Configuration Tool; eCos Minimal  
  Configuration file (.ecm)  
ecos.db (database) file, 10, 227, 240, 264–66  
  example, 266  
  modifying, 324–25, 330  
ecosadmin.tcl (administration) file, 275  
EL/IX. *See* compatibility layers; POSIX;  
  EL/IX  
evaluation boards, 337–44  
exceptions, 31–40  
  example, 36–38  
  HAL macros, 38–39  
  handling, 32–35  
    application layer, 39  
    default handler, 34–35  
  kernel API, 35–36

## F

file systems, 12, 155–60  
  Journalling Flash File System  
    Version 2 (JFFS2), 160  
  POSIX file I/O, 155–56  
  RAM, 158–60  
  ROM, 157–58  
flags, 110–113  
  example, 112–13  
  kernel API, 110–12  
FreeBSD. *See* networking; FreeBSD

## G

GDB stub, 16, 35, 154–55, 186, 197  
  configuration, 154  
Gilmore, John, 1  
GNU Binary Utilities (binutils), 221  
  building, 361–68  
GNU C/C++ Compiler (GCC), 1, 221,  
  301, 316  
  building, 361–68

GNU cross development tools  
  building, 361–68  
  installation, 220–23  
  mailing lists, 221–22  
  versions, 220–21  
GNU Debugger (GDB), 1, 221–22  
  building, 361–68  
  Command Line Interface (CLI), 309  
  compiler optimization, 305  
  debugging applications, 305–9  
  protocol, 69  
  running without Insight interface, 306  
GNU linker (ld), 4, 258, 301, 331  
GNU make, 281, 299–302  
GNU Public License (GPL), 208, 345–53  
GoAhead WebServer, 180–82  
GPL. *See* GNU Public License (GPL)

## H

HAL. *See* Hardware Abstraction Layer (HAL)  
Hardware Abstraction Layer (HAL), 5, 12,  
  17–29, 65, 316  
  clocks, 122  
  configuration, 24–26  
  directory structure, 19–22  
  exceptions, 32–35, 38–39  
  interrupts, 42–44, 50–52, 53–56  
  macros, 23–24  
  porting, 315–35  
  stacks, 94  
  startup, 27–29  
  submodules, 19  
Henkel-Wallace, David, 1

## I

I/O control system, 67, 140–47  
  device drivers. *See* device drivers  
I/O sub-system, 142–46  
  configuration, 143  
  device I/O table, 142–43, 146  
  example, 145–46  
  kernel API, 143–44  
In-Circuit Emulator (ICE), 317

- Insight, 221, 305–9, 311–13. *See also* GNU Debugger (GDB)
- Interrupt Service Routine (ISR), 40–41, 43, 45–47, 68, 146–47  
 example, 48–50  
 management, 51–53
- interrupts, 40–58, 77  
 configuration, 42–44  
 handling, 44–47  
 kernel API, 52–58  
 management, 50–58  
 scheduler synchronization, 41  
 stack, 43, 94–95
- ISO C library. *See* libraries; ISO C
- ISR. *See* Interrupt Service Routine (ISR)
- J**
- JFFS2. *See* Journalling Flash File System Version 2 (JFFS2)
- K**
- kernel, 12, 15, 73–84, 122–24, 183  
 C API, 73–74  
 directory structure, 74–75  
 instrumentation, 74  
 scheduler, 41, 77–84, 183  
 bitmap, 81  
 configuration, 83–84  
 kernel API, 78–79  
 locking/unlocking, 45, 47, 77–78  
 multi-level queue, 79–81  
 priority levels, 81–83  
 startup, 75–76  
 timeslicing, 79–81, 84, 123
- L**
- libraries, 12, 15  
 ISO C, 77, 138–40  
 math, 138–40
- libtarget.a (eCos library) file, 281, 296, 299, 302
- license 2. *See also* GNU Public License (GPL)
- linker flags, 284–85
- linker script files (.ld), 34, 60, 257–60, 302, 331–32
- lint. *See* Splint
- Linux, 5, 151, 208
- Load Memory Address (LMA), 260
- lwIP. *See* networking; lwIP
- M**
- mailboxes. *See* message boxes
- mailing lists, 6–7
- makefile, 282–84, 300–2
- math library. *See* libraries; math
- Memory Layout Tool (MLT), 223, 248, 257–64
- memory manipulation, 257–64
- message boxes, 113–18  
 configuration, 113–14  
 example, 117–18  
 kernel API, 114–16
- MIB. *See* Simple Network Management Protocol (SNMP)
- Microwindows, 184
- MLT. *See* Memory Layout Tool (MLT)
- multi-level queue scheduler. *See* kernel; scheduler
- mutexes, 95–101  
 configuration, 97  
 example, 99–101  
 kernel API, 97–99  
 priorities, 96
- N**
- networking, 13, 15, 167–79  
 Basic Networking Framework, 168, 170, 171–73, 176, 310  
 BOOTP. *See* Bootstrap Protocol (BOOTP)  
 configuration, 171–76  
 DHCP. *See* Dynamic Host Configuration Protocol (DHCP)  
 Domain Name System (DNS), 178–79  
 Ethernet initialization, 175  
 FreeBSD, 15, 169–70, 173, 176  
 lwIP, 170

OpenBSD, 168, 170, 173, 176  
tests, 172, 174, 176–78  
threads, 170–71  
NEWS file (latest eCos information), 149

## O

objcopy utility (binutils), 299, 302–3, 328  
online repository, 12, 229. *See also* WinCVS  
OpenBSD. *See* networking; OpenBSD  
options. *See* configuration options

## P

Package Administration Tool, 9, 10, 275–77  
    building, 277  
    installation, 228  
packages, 8, 10, 14, 239–47, 253  
    adding and removing, 274  
    Component Definition Language. *See*  
        Component Definition Language  
        (CDL)  
    directory structure, 240–41  
    distribution files (.epk), 276–77  
Peripheral Component Interconnect (PCI)  
    bus, 160–65  
platform. *See* Hardware Abstraction Layer  
    (HAL); submodules  
platform support. *See* evaluation boards  
porting, 315–35  
    hints, 334  
POSIX. *See* compatibility layers; POSIX  
power management, 184  
priority levels. *See* threads; priority levels  
processor support, 6, 337–44  
properties, 9. *See also* Configuration Tool;  
    Properties window

## R

Real-Time Clock (RTC). *See* clocks;  
    Real-Time Clock (RTC)  
RedBoot, 13, 15, 59, 153, 176, 185–206,  
    317, 327  
    eCos Minimal Configuration file (.ecm),  
        188, 290

binary images, 237  
boot scripting, 204–6  
booting, 293–95  
building, 286–87, 288–93  
Command Line Interface (CLI), 196  
commands, 196–204  
configuration, 189–93  
directory structure, 187–88  
GDB stub, 197, 202  
host configuration, 193–95, 286–88  
installation, 292–95  
IP addresses, 194, 287  
loading applications, 303–5  
telnet, 194–95, 295  
repository database, 264, 325  
ROM monitors 152–55. *See also* RedBoot

## S

scheduler. *See* kernel; scheduler  
semaphores, 101–5  
    example, 47–50, 103–5, 300  
    kernel API, 101–3  
Simple Network Management Protocol  
    (SNMP), 13, 179–80  
    disadvantages, 180  
    Management Information Base (MIB), 179  
    UCD-SNMP, 13, 179  
simulators, 21–22, 311–13  
size utility (binutils), 303  
SMP. *See* Symmetric Multi-Processing (SMP)  
Source Navigator, 2, 278–79  
spinlocks 118–20. *See also* Symmetric  
    Multi-Processing (SMP)  
Splint, 279  
stacks. *See* threads; stacks  
Startup Type (configuration option),  
    26, 257–58, 270, 297  
stub. *See* GDB stub  
suboptions, *See* configuration options;  
    suboptions  
Symmetric Multi-Processing (SMP),  
    57–58, 79, 118, 182–83  
synchronization mechanisms, 95–120  
    blocking/nonblocking, 95

**T**

target.ld file, 299, 302. *See also* linker script files (.ld)  
targets, 14–15  
Tcl. *See* Tool Command Language (Tcl)  
telnet. *See* RedBoot; telnet  
templates, 15–16, 26, 252, 264, 266, 270–72  
    adding, 325–26  
    example, 290, 297  
terminology, 8–16  
tests, 6, 242, 310–11. *See also*  
    networking; tests  
threads, 40–41, 43, 47, 85–95. *See also*  
    networking; threads  
    configuration, 86–87  
    example, 92–94, 300  
    kernel API, 87–92  
    priority levels, 81–83, 96–97  
    stacks, 94–95  
    startup, 77  
Tiemann, Michael, 1  
timers, 86, 133  
TkCVS, 230  
Tool Command Language (Tcl),  
    243, 275, 278  
tracing. *See* asserts and tracing

**U**

UCD-SNMP. *See* Simple Network Management Protocol (SNMP)  
Universal Serial Bus (USB), 165–67

**V**

variant. *See* Hardware Abstraction Layer (HAL); submodules  
Vector Service Routine (VSR),  
    33–35, 38–39, 44, 45–47  
    kernel API, 39–40  
    table, 33–34, 38–39, 44, 61  
Virtual Memory Address (VMA), 260  
Virtual Vector Table (VVT),  
    60–67, 191  
    initialization, 64–65  
virtual vectors, 59–71  
    configuration, 63–64  
    ROM monitors, 63  
VSR. *See* Vector Service Routine (VSR)  
VVT. *See* Virtual Vector Table (VVT)

**W**

web-based management. *See* GoAhead WebServer  
WinCVS, 230–38, 240  
Windows, 5, 208–9

**Z**

zlib, 183

# The GNU General Public License (GPL)

## Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.  
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making



the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## **TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

**0.** This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

**1.** You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

**2.** You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy

of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

**3.** You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the

major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

**4.** You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

**5.** You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

**6.** Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

**7.** If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license

practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

**8.** If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

**9.** The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

**10.** If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### **NO WARRANTY**

**11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**12.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY

MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

### **How to Apply These Terms to Your New Programs**

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free

software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest  
in the program 'Gnomovision' (which makes passes at compilers)  
written by James Hacker.

signature of Ty Coon, 1 April 1989  
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

## A B O U T T H E C D - R O M

The CD-ROM included with *Embedded Software Development with eCos* contains all of the software and source code needed to set up a complete embedded software development environment, centered on the eCos real-time operating system. The CD-ROM includes the following:

- A readme file, *Readme.txt*, which gives additional details about the contents of the CD-ROM and installation of the software it contains.
- A UNIX environment for Windows consisting of a Dynamic Link Library (DLL) that acts as a UNIX emulation layer and a collection of UNIX tools ported to Windows called Cygwin.
- A snapshot of the eCos source code repository along with all of the eCos configuration and development tools.
- Example files demonstrating how to use the eCos configuration tools to build the RedBoot™ ROM monitor, the eCos library, and an application. As well as an example showing how to port the eCos HAL.
- The GNU cross-development tools for the i386 and PowerPC platforms including the GNU Binary Utilities (binutils), GNU C/C++ Compiler (GCC), and GNU Debugger (GDB) with the Insight graphical user interface.
- An open-source lint program, named Splint, which is used to statically verify a program, or part of a program, against standard libraries.
- An open-source code analysis and comprehension tool named Source-Navigator.
- A Windows-based CVS client called WinCVS, which allows anonymous access to the eCos online source code repository.
- A popular Linux CVS client called TkCVS.

The CD-ROM can be used on Microsoft Windows® 95/98/NT®/2000/Me/XP and Linux. The maintainers have tested the eCos tools on Windows NT 4.0 (with service pack 3 or above) and Red Hat Linux 7.0 (or later). The Cygwin tools are for use on Windows-based host systems.

The recommended operating system, which is used for the examples in this book, is Windows NT with service pack 6.0a.

### *License Agreement*

Use of the software accompanying *Embedded Software Development with eCos* is subject to the terms of the License Agreement and Limited Warranty, found on the previous seven pages.

### *Technical Support*

Prentice Hall does not offer technical support for any of the programs on the CD-ROM. However, if the CD-ROM is damaged, you may obtain a replacement copy by sending an email that describes the problem to: [disc\\_exchange@prenhall.com](mailto:disc_exchange@prenhall.com).