eCosPro User Guide

eCosPro User Guide

Publication date 10 January 2025
Copyright © 2001-2004, 2009, 2010-2011 Free Software Foundation, Inc.
Copyright © 2010-2011, 2015-2024 eCosCentric Limited.

Open Publication License

The document containing or referencing this license was produced in full, or in part if the document contains multiple licensing references, from work that is 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/).

Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder(s).

eCosPro Non-Commercial Public License

The document containing or referencing these licenses was produced in full, or in part if the document contains multiple licensing references, from work that is subject to the terms and conditions of the eCosPro Non-Commercial Public License.

Distribution of the work or derivative of the work is permitted for Non-Commercial[*] use only.

[*]As defined by the eCosPro Non-Commercial Public License.

Trademarks

Altera® and Excalibur™ are trademarks of Altera Corporation.

AMD® is a registered trademark of Advanced Micro Devices, Inc.

ARM®, Cortex-M®, StrongARM®, Thumb®, ARM7™, ARM9™ are trademarks of Advanced RISC Machines, Ltd.

Cirrus Logic® and Maverick™ are registered trademarks of Cirrus Logic, Inc.

Cogent™ is a trademark of Cogent Computer Systems, Inc.

Compaq® is a registered trademark of the Compaq Computer Corporation.

Debian® is registered trademark of Software in the Public Interest, Inc.

eCos®, eCosCentric® and eCosPro® are registered trademarks of eCosCentric Limited.

Fujitsu® is a registered trademark of Fujitsu Limited.

IBM®, and PowerPC™ are trademarks of International Business Machines Corporation.

IDT® is a registered trademark of Integrated Device Technology Inc.

Intel®, i386™, Pentium®, StrataFlash® and XScale™ are trademarks of Intel Corporation.

Intrinsyc® and Cerf™ are trademarks of Intrinsyc Software, Inc.

Linux® is a registered trademark of Linus Torvalds.

Matsushita™ and Panasonic® are trademarks of the Matsushita Electric Industrial Corporation.

Microsoft®, Windows®, Windows NT®, Windows XP® and Windows 7® are registered trademarks of Microsoft Corporation, Inc.

MIPS®, MIPS32™ MIPS64™, 4K™, 5K™ Atlas™ and Malta™ are trademarks of MIPS Technologies, Inc.

Motorola® and ColdFire® are trademarks of Motorola, Inc.

NEC®, V800™, V850™, V850/SA1™, V850/SB1™, VR4300™ and VRC4375™ are trademarks of NEC Corporation.

openSUSE™, is a trademark of Novell, Inc. in the US and other countries

PMC-Sierra®, RM7000™ and Ocelot™ are trademarks of PMC-Sierra Incorporated.

Red Hat®, Fedora™, RedBoot™, GNUPro® and Insight™ are trademarks of Red Hat, Inc.

Samsung® and CalmRISC™ are trademarks or registered trademarks of Samsung, Inc.

Table of Contents

List of Figures

List of Tables

List of Examples

Part I. Introduction

Table of Contents

Chapter 1. Key Features

- eCos® is distributed under the GPL license with an exception which permits proprietary application code to be linked with eCos without requiring that it also falls under the GPL. It is royalty and buyout free.

- eCosPro® extends the functionality of eCos to provide a rich set of features as well as additional platform and device drivers support. eCosPro features as well as additional platforms and device drivers can be found on the eCosCentric web site.

- eCosPro Developer's Kits combine the stability, feature set, and quality of support required for commercial embedded application development with eCos by professional developers of embedded hardware devices.

- As an Open Source project, eCos is under constant improvement, with an active developer community, based around the eCos community web site at http://ecos.sourceware.org/.

- Powerful GUI-based configuration system allowing both large and fine grained configuration of eCos. This allows the functionality of eCos to be customized to the exact requirements of the application.

- Full-featured, flexible, configurable, real time embedded kernel. The kernel provides thread scheduling, synchronization, timer, and communication primitives. It handles hardware resources such as interrupts, exceptions, memory and caches.

- The Hardware Abstraction Layer (HAL) hides the specific features of each supported CPU and platform, so that the kernel and other run-time components can be implemented in a portable fashion.

- Support for µITRON and POSIX Application Programmer Interfaces (APIs). It also includes a fully featured, thread-safe ISO standard C library and math library.

- eCosPro Developer's Kits include additional features, such Symmetric Multi-Processing (SMP) for ARM and Cortex-A, as well as eCos libraries and packages, such as full standard C++ library support, BootUp, Robust Boot Loader (RBL), FlashSafe API for high-reliability, data logging, additional filesystems (multi-threaded FAT, JFFS, MMFS, YAFFS) and more.

- eCosPro Developer's Kits also provide Eclipse-based IDE (Linux or Windows based), with eCos embedded-target support.

- Support for a wide variety of devices including many serial devices, ethernet controllers and FLASH memories. There is also support for PCMCIA, USB and PCI interconnects.

- A fully featured TCP/IP stack implementing IP, IPv6, ICMP, UDP and TCP over ethernet, as well as a lightweight IP stack (lwIP) designed for embedded systems. Support for SNMP, HTTP, TFTP and FTP are also present.

- The RedBoot ROM monitor is an application that uses the eCos HAL for portability. It provides serial and ethernet based booting and debug services during development.

- Many components include test programs that validate the components behaviour. These can be used both to check that hardware is functioning correctly, and as examples of eCos usage.

- eCos and eCosPro documentation include this User Guide, an Installation Guide, the Reference Manual and the Components Writer's Guide.

Chapter 2. eCos Overview

eCos® is an open source, configurable, portable, and royalty-free embedded real-time operating system. The following text expands on these core aspects that define eCos.

eCos is provided as an open source runtime system supported by the GNU open source development tools. Developers have full and unfettered access to all aspects of the runtime system. No parts of it are proprietary or hidden, and you are at liberty to examine, add to, and modify the code as you deem necessary. These rights are granted to you and protected by the GNU Public License (GPL). An exception clause has been added to the eCos license which limits the circumstances in which the license applies to other code when used in conjunction with eCos. This exception grants you the right to freely develop and distribute applications based on eCos. You are not expected or required to make your embedded applications or any additional components that you develop freely available so long as they are not derived from eCos code. We of course welcome all contributions back to eCos such as board ports, device drivers and other components, as this helps the growth and development of eCos, and is of benefit to the entire eCos community. See the section called " eCos Licence Overview" for more details.

One of the key technological innovations in eCos is the configuration system. The configuration system allows the application writer to impose their requirements on the run-time components, both in terms of their functionality and implementation, whereas traditionally the operating system has constrained the application's own implementation. Essentially, this enables eCos developers to create their own application-specific operating system and makes eCos suitable for a wide range of embedded uses. Configuration also ensures that the resource footprint of eCos is minimized as all unnecessary functionality and features are removed. The configuration system also presents eCos as a component architecture. This provides a standardized mechanism for component suppliers to extend the functionality of eCos and allows applications to be built from a wide set of optional configurable run-time components. Components can be provided from a variety of sources including: the standard eCos release; commercial third party developers or open source contributors.

The royalty-free nature of eCos means that you can develop and deploy your application using the standard eCos release without incurring any royalty charges. In addition, there are no up-front license charges for the eCos runtime source code and associated tools.

eCos is designed to be portable to a wide range of target architectures and target platforms including 16, 32, and 64 bit architectures, MPUs, MCUs and DSPs. The eCos kernel, libraries and runtime components are layered on the Hardware Abstraction Layer (HAL), and thus will run on any target once the HAL and relevant device drivers have been ported to the target's processor architecture and board. Currently eCos supports a large range of different target architectures:

• ARM7, ARM9, Cortex-A5, Cortex-A9, Cortex-M3, Cortex-M4, Cortex-M7, Intel StrongARM and XScale.

• Fujitsu FR-V

• Hitachi SH2/3/4

• Hitachi H8/300H

• Intel x86

• MIPS

• Matsushita AM3x

• Motorola PowerPC

• Motorola 68k/Coldfire

• NIOS II

• NEC V850

- Sun SPARC

- SuperH

- Tilera TILE-Gx

including many of the popular variants of these architectures and evaluation boards. For a current list of commercially supported hardware see the eCosPro CPU & Board Support pages.

eCos has been designed to support applications with real-time requirements, providing features such as full preemptability, minimal interrupt latencies, and all the necessary synchronization primitives, scheduling policies, and interrupt handling mechanisms needed for these type of applications. eCos also provides all the functionality required for general embedded application support including device drivers, memory management, exception handling, C, math libraries, etc. In addition to runtime support, the eCos system includes all the tools necessary to develop embedded applications, including eCos software configuration and build tools, and GNU based compilers, assemblers, linkers, debuggers, and simulators.

All eCosPro® Developer's Kits are backed by commercial support provided by members of the original team which developed eCos. Support is provided through the eCosPro issue management system.

The latest and most complete set of eCosPro documentation and installation guides may be found online at https://doc.ecoscentric.com/index.html.

In addition, you may wish to visit the eCos open source developers site: http://ecos.sourceware.org/. The site is dedicated to the eCos developer community and contains news, FAQ, discussion and announcement mailing lists, among other items.

eCos and eCosPro are released as open source software because the eCos maintainers and eCosCentric believe that this is the most effective software development model, and that it provides the greatest benefit to the embedded developers.

Chapter 3. Licensing

eCosPro Non-Commercial Public Licence Overview

The eCosPro Non-Commercial Public License applies to elements of eCosCentric's intellectual property within a distribution. Depending on the particular release this may cover various files within the HAL, device drivers, documentation and other components. All these files are clearly marked with a reference to the eCosPro Non-Commercial Public Licence in their headers. Key requirements of the license are:

- You may **only** make non-commercial use of these elements, as well as any of their derivatives in any form. Non-commercial use includes the distribution of all formats of these elements.

- Commercial use and/or distribution **requires** an eCosPro License

For further clarification of non-commercial use, refer to the Definitions section of the license and its Example Usage appendix.

When distributing code, or derivatives such as object or binary code, under this license, including any hardware or device containing the code or its derivatives, you MUST:

- give notice that portions of the software, within the device if applicable, is covered by this license and may NOT be used for commercial purposes; and

- include the text of and/or the URI to, this license; and

- either (a) include the software source with the distribution or device in machine readable format on a physical medium; or (b) include a note either offering the recipient the sofware source in machine readable format on a physical medium or inform the recipient where the software sources may be accessed from, if necessary providing the server connected to the internet from where the software source may be accessed.

eCosPro Licence Overview

The eCosPro License removes all of the requirements of the eCosPro Non-Commercial Public License, permitting distribution of binary and object code deriatives without restriction and allowing commercial use of the software. A commercial eCosPro Distribution also includes files within the HAL, device drivers, and other components which are clearly marked with a reference to the eCosPro License in their headers. While binary and object code derivatives of these files may be freely copied, these source files themselves must not be redistributed.

- You CANNOT distribute any source code provided exclusively under the eCosPro License.

- Source code provided originally under the eCosPro Non-Commercial Public License and subsequently re-licensed under the eCosPro License does not have to be made available to recipients of binary or object code derivatives. However, you may still choose to make such source code available but may ONLY do so under the under the terms and conditions of the eCosPro Non-Commercial Public License.

- Although not specifically an eCosPro license issue, you MUST also ensure that you do not redistribute any eCosCentric proprietary applications or information. This includes your licensee file, if provided.

The full eCosPro License is available online at https://www.ecoscentric.com/licensing/ecospro-license.shtml.

When shipping a product with eCos or eCosPro, developers and OEMs are encouraged to read the eCosPro Software Licenses Compliance Guide which provides a set of guidelines to follow when releasing products that incorporate software from an eCosPro release. It does not claim to be authoritative, it is still your responsibility to read, understand and comply with the various licenses that apply to different portions of the runtime source code.

eCos Licence Overview

On May 2002, eCos was released under a modified version of the well known GNU General Public License (GPL), making it an official GPL- compatible Free Software License. An exception clause has been added to the eCos license which limits the circumstances in which the license applies to other code when used in conjunction with eCos. The exception clause is as follows:

```
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.
```

The goal of this license is to serve the eCos user community as a whole. It allows all free eCos users to develop products without paying anybody anything, no matter how many developers are working on the product or how many units will be shipped. The license also guarantees that the eCos source code will always be freely available. This applies not only to the core eCos code itself but also to any changes that anybody makes to the core. In particular, it should prevent any company or individual contributing code to the system and then later claiming that all eCos users are now guilty of copyright or patent infringements and have to pay royalties. It should also prevent any company from making some small improvements, calling the result a completely new system, and releasing this under a new and less generous license.

The license does *not* require developers to release the source code of any *applications* that includes eCos. However, if you make any changes to code covered by the eCos license, or writes new files derived in any way from eCos code, then the license stipulates that these changes must be made available in source code form to all recipients of binaries based on the modified code, either by including the sources along with the binaries they deliver (or with any device containing such binaries) or with a written offer to supply the source code to the general public for three years. eCosPro Non-Commercial Public licensed code has similar requirements.

eCosPro licensed code is naturally excluded from this requirement.

It is perhaps most practical for eCos developers to make the source code available online and inform those who are receiving binaries containing eCos code, and probably also the eCos maintainers, about the location of the code. See the full text of the GPL for the most authoritative definition of the obligations.

Although it is not strictly necessary to contribute the modified code back to the eCos open source project, the eCos maintainers are always pleased to receive code contributions and are responsible for deciding whether such contributions should be applied to the public repository. In addition, a copyright assignment is required for any significant changes to the core eCos packages.

The result is a royalty-free system with minimal obligations on the part of eCos application developers. This has resulted in the rapid uptake of eCos. At the same time, eCos and eCosPro are fully open source with all the benefits that implies in terms of quality and innovation providing a winning combination.

Questions and answers

The following queries provide some clarification as to the implications of the eCos license. They do not consititute part of the legal meaning of the license.

Q. What is the effect of the eCos license?

A. In the simplest terms, when you distribute anything containing eCos code, you must make the source code to eCos available under the terms of the GPL.

Q. What if I make changes to eCos, or write new code based on eCos code?

A. Then you must make those changes available as well.

Q. Do I have to distribute the source code to my application? Isn't the GPL "viral"?

A. You do not have to distribute any code under the terms of the GPL other than eCos code or code derived from eCos. For example, if you write a HAL port based on copying an existing eCos HAL in any way, you must make the source code available with the binary. However you would not need to make available any other code, such as the code of a wholly separate application linked with eCos. For guidance see the eCosPro Software Licenses Compliance Guide.

Q. I would rather stick with the RHEPL code, but I updated my anonymous CVS checkout.

A. You can check out the final version of anonymous CVS before the license change using the CVS tag `last-rhepl`. See the anonymous CVS access page for details.

Previous eCos License

Prior to May 2002, eCos was released under the Red Hat eCos Public License (RHEPL). The RHEPL required any modifications to eCos code to be made available under preferential terms to Red Hat and was therefore incompatible with code licensed under the GPL. The use of eCos source code which was licensed under the RHEPL is not affected by the switch to the modified GPL for later revisions.

Documentation License

Most of the HTML and online eCosPro documentation is published under one of three licenses:

- The Open Publication License; or

- The eCosPro Non-Commercial Public License

- The eCosPro License

The PDF version of the documentation is a collection of these works and is therefore also provided under both of these licenses. The original SGML from which items (partitions, chapters, sections, etc) licensed under the Open Publication License may be downloaded from http://ecos.sourceware.org/ or is provided as part of your installation of eCosPro. The full text of the Open Publication License may be found at http://www.opencontent.org/openpub/. Modification, reproduction or distribution of any documentation published under the eCosPro License in any form is strictly prohibited, while documentation generated from source or derivatives licensed under the eCosPro Non-Commercial Public License may only be published for non-commercial purposes and carries with it requirements to make the sources or derivatives available to its recipients.

HTML and online versions of the documentation each contain a link at the bottom of each page that specifies the license under which it was provided.

Chapter 4. Notation and Conventions

Since there are many supported target architectures, notation conventions are used in this manual to avoid repeating instructions that are very similar.

GDB and GCC Command Notation

Cross-development commands like **gcc** and **gdb** will be shown with a *TARGET-* prefix. You need to replace *TARGET-* with the correct prefix before using the command. Just using **gcc** or **gdb** will use the tools for the host, which is not (usually) what you want.

For example use **arm-eabi-gcc** and **arm-eabi-gdb** for ARM7, ARM9, Thumb, Cortex-A, and Cortex-M targets. Use **powerpc-eabi-gcc** and **powerpc-eabi- gdb** for PowerPC targets. Use **i386 -elf-gcc** and **i386-elf-gdb** for IA32 targets. And so on, the exact prefix to use is shown in the documentation for each target.

Note that some versions of the GCC cross compiler generate executable files with the `.exe` suffix on Windows, but not on Linux. The suffix `.exe` will be omitted from executable file names, so you will see `hello` instead of `hello.exe`.

Directory and File System Conventions

The default directory for eCosPro distributions on Windows (usually `C:\eCosPro`) is different from that on Linux (usually `/opt/ecospro` or `$HOME/ecospro` for non-root installations). Since many command line examples in the tutorials use these paths, this default (base) directory will be cited as *BASE_DIR*.

Windows and Linux have a similar file system syntax, but the MS-DOS command interpreter on Windows versions prior to Windows 10 uses the backslash character (\) as a path separator, while Linux and POSIX shells (including the bash shell for windows) use the forward slash (/). Windows 10 can use both characters.

This document will use the POSIX shell convention of forward slashes throughout.

Version Conventions

This manual does not refer explicitly to any particular distribution of eCos or eCosPro. However, version numbers form part of many file path names. In all of these places the version number will be shown like this: *<version>*.

If you are using the freely available public distribution of eCos at http://hg-pub.ecoscentric.com/ and have used Mercurial to check eCos out of the repository, the version number will always be `current`, since that is the name of the directory in the repository. When a stable release is made this directory name is changed, in the release, to the number of the release, for example `v3_1_71` or `v4_0_3`.

Chapter 5. Documentation Roadmap

The eCos and eCosPro documentation is divided into a number of components:

eCosPro Developer's Kits: Installation and Getting Started Guide

> This document describes how to install the eCosPro Developer's Kit and get started with eCosPro.

User Guide

> This document. It includes the following sections:

Programming With eCos	This section describes how to write programs that run under eCos by running through some examples.
The eCos Configuration Tool	This section describes the eCos graphical configuration tool and how to use it to change how eCos behaves.
eCos Programming Concepts and Techniques	An explanation of the eCos programming cycle, and a description of some debugging facilities that eCos offers.
Configuration and the Package Repository	Information on how to configure eCos manually, including a reference on the **ecosconfig** command, memory layouts, and information on how to manage a package repository using the eCos Package Administration Tool.

eCosPro Reference Guide

> The eCosPro Reference Guide provides detailed documentation on various aspects of eCos and eCosPro and well as descriptions of hardware support, board setup and timings. It also incorporates other guides such as the RedBoot User's Guide and the eCosPro-SecureSockets User's Guide. This document is being constantly updated online at https://doc.ecoscentric.com/ref/index.html, so the following list just mentions the more important sections, take a look at the guide itself for the full story. A copy of the eCosPro Reference Guide is also provided with each eCosPro Developer's Kit in both HTML and PDF format.

> The *eCos Kernel*

>> In-depth description of eCos's native C kernel API. Important considerations are given for programming the eCos kernel. The semantics for each kernel function are described, including how they are affected by configuration.

> *The eCos Hardware Abstraction Layer (HAL)*

>> A description of the structure and functionality of the eCos HAL. This section also includes a porting guide to help moving eCos to different platforms.

> *I/O Packages (Device Drivers)*

>> A description of the philosophy behind eCos device drivers, as well as a presentation of the C language APIs for using the current device drivers.

>> Device driver support includes serial, ethernet and FLASH devices, and support for PCI, PCMCIA and USB interconnects.

> *POSIX and µITRON APIs*

>> A description of the POSIX and µITRON APIs and how they are supported under eCos.

> *File System Support Infrastructure*

>> This describes the filesystem infrastructure provided in eCos. This is implemented by the FILEIO package and provides POSIX compliant file and IO operations together with the BSD socket API.

Other components, such as:

- FAT

- Multimedia

- YAFFS

- JFFS2 filesystems are also described.

TCP/IP Stack Support for eCos

This describes the Common Networking for eCos package, which provides support for a complete TCP/IP networking stack. The design allows for the actual stack to be modular permitting different implementations. Currently only a version based on FreeBSD is are available as the earlier version (circa 2000) based on OpenBSD was deprecated and has now been removed from current eCosPro releases.

Other components related to networking, including support for SNMP, DNS, HTTP and FTP, are also described.

eCosPro-SecureSockets User Guide

This provides a guide to the eCos OpenSSL port, an optional add-on package for users of the eCosPro Developer's Kit.

Architectural and Platform Support

This describes eCosPro support for the different architectures along with support for individual platforms, platform configuration, HAL support (including debug, profiling, diagnostic and SMP support), board setup and startup.

RedBoot User's Guide

This describes RedBoot, which provides a complete bootstrap environment for a range of embedded operating systems, such as embedded Linux and eCos, and includes facilities such as network downloading and debugging. It also provides a simple flash file system for boot images.

Component Writer's Guide

The Component Writer's Guide is intended for developers who need to add or modify parts of eCos itself. It describes the following things:

Overview   An explanation of the configuration technology used in eCos, why it is done this way, how it works and the terminology used.

Package Organization A description of the eCos package repository, how it is organized and how packages themselves are organized.

The CDL Language   A description of the CDL language and how it is used to control the configuration of eCos components. The document also contains a complete specification of the language.

The Build Process   A description of what happens once a configuration has been created and must be built into a set of executables.

eCosPro CDT plugin user's guide

This provides documentation on the eCosPro CDT plugin as well as Eclipse/CDT usage notes for application developers using the eCosPro Developer's Kit.

Part II. Programming With eCos

Table of Contents

Chapter 6. Programming With eCos

The following chapters of this manual comprise a simple tutorial for configuring and building eCos, building and running eCos tests, and finally building three stand-alone example programs which use the eCos API to perform some simple tasks.

You will need a properly installed eCos system, with the correct versions of the GNU toolchain, Host Tools and Eclipse. The eCosPro installer will perform all of the necessary configuration during installation for you to allow you to build and run applications on the hardware supported by the eCosPro Developer's Kit out-of-the-box.

eCosPro developers will typically use the Eclipse development environment and eCosPro Eclipse plugin provided as part of their eCosPro Developer's Kit installation.

Alternatively, you can also integrate a command line environment with whatever editor and revision control system you are familiar with. For the command line environment, on Linux this will typically be a command shell through a console or terminal window. On Windows you can use either the Command Prompt (**CMD.exe**) or the bash command line interpreter provided with your eCosPro installation, with the environment variables set as described in the section called "Environment Variables" and the chapter Environment Variables in the eCosPro Developer's Kit - Installation and Getting Started Guide.

The Development Process

Most development projects using eCos would contain some (or most) of the following:

eCos Configuration

eCos is configured to provide the desired API (the inclusion of libc, uitron, and the disabling of certain undesired funtions, etc.), and semantics (selecting scheduler, mutex behavior, etc.). See Chapter 7, *Configuring and Building eCos from Source*.

It would normally make sense to enable eCos assertion checking at this time as well, to catch as many programming errors during the development phase as possible.

Note that it should not be necessary to spend much time on eCos configuration initially. It may be important to perform fine tuning to reduce the memory footprint and to improve performance later when the product reaches a testable state.

Integrity check of the eCos configuration

While we strive to thoroughly test eCos, the vast number of configuration permutations mean that the particular configuration parameters used for your project may not have been tested. Therefore, we advise running the eCos tests after the project's eCos configuration has been determined. See Chapter 8, *Running an eCos Test Case*.

Obviously, this should be repeated if the configuration changes later on in the development process.

Application Development - Target Neutral Part

While your project is probably targeting a specific architecture and platform, possibly custom hardware, it may be possible to perform part of the application development using simulated or synthetic targets.

There are three good reasons for doing this:

• It may be possible by this means to perform application development in parallel with the design/implementation of the target hardware, thus providing more time for developing and testing functionality, and reducing time-to-market.

• The build-run-debug-cycle may be faster when the application does not have to be downloaded to a target via a serial interface. Debugging is also likely to be more responsive when you do not have to to communicate with the remote GDB stubs in RedBoot via serial. It also removes the need for manually or automatically resetting the target hardware.

- New hardware can often be buggy. Comparing the behaviour of the program on the hardware and in the simulator or synthetic target may allow you to identify where the problems lie.

This approach is possible because all targets (including simulators and synthetic ones) provide the same basic API: that is, kernel, libc, libm, uitron, infra, and to some extent, HAL and IO.

Synthetic targets are especially suitable as they allow you to construct simulations of elaborate devices by interaction with the host system, where an IO device API can hide the details from the application. When switching to hardware later in the development cycle, the IO driver is properly implemented.

Simulators can also do this, but it all depends on the design and capabilities of the simulator you use. Some like Bochs provide complete hardware emulation, while others just support enough of the instruction set to run compiled code.

Therefore, select a simulator or synthetic target and use it for as long as possible for application development. That is, configure for the selected target, build eCos, build the application and link with eCos, run and debug. Repeat the latter two steps until you are happy with it.

Obviously, at some time you will have to switch to the intended target hardware, for example when adding target specific feature support, for memory footprint/performance characterization, and for final tuning of eCos and the application.

Application Development - Target Specific Part

Repeat the build-run-debug-cycle while performing final tuning and debugging of application. Remember to disable eCos assertion checking if you are testing any performance-related aspects, and to enable compiler optimization (e.g. -O2) it can make a big difference.

It may be useful to switch between this and the previous step repeatedly through the development process; use the simulator/synthetic target for actual development, and use the target hardware to continually check memory footprint and performance. There should be little cost in switching between the two targets when using two separate build trees.

Chapter 7. Configuring and Building eCos from Source

This chapter documents the configuration of eCos. The process is the same for any of the supported targets: you may select a hardware target (if you have a board available), any one of the simulators, or a synthetic target (if your host platform has synthetic target support).

eCos Start-up Configurations

There are various ways to download an executable image to a target board, and these involve different ways of preparing the executable image. In the eCos Hardware Abstraction Layer (HAL package) there are configuration options to support the different download methods. Table 7.1, "Configuration for various download methods" summarizes the ways in which an eCos image can be prepared for different types of download. This is not an exhaustive list, some targets define additional start-up types of their own. Where a ROM Monitor is mentioned, this will usually be RedBoot, although on some older, or low resource, targets you may need to use the GDB stubs ROM or BootUp, see the target documentation for details.

Table 7.1. Configuration for various download methods

Download method	HAL configuration
Burn hardware ROM	ROM, ROMRAM or JTAG start-up
Download to ROM emulator	ROM or ROMRAM start-up
Download to board with ROM Monitor	RAM start-up
Download to simulator without ROM Monitor	ROM start-up
Download to simulator with ROM Monitor	RAM start-up
Download to simulator ignoring devices	SIM configuration
Run synthetic target	RAM start-up

> **⚠ Caution**
>
> You cannot run an application configured for RAM start-up on the simulator directly: it will fail during start-up. You can only download it to the simulator if you are already running RedBoot in the simulator, as described in the toolchain documentation or you load through the *SID* GDB debugging component. This is not the same as the simulated stub, since it does not require a target program to be running to get GDB to talk to it. It can be done before letting the simulator run or you use the ELF loader component to get a program into memory.

> **📝 Note**
>
> • Only the Linux Synthetic target simulator and TILE-Gx simulators are supported by eCosPro, although customer-specific simulator support is available from eCosCentric.
>
> • Configuring eCos' HAL package for simulation should rarely be needed for real development; binaries built with such a kernel will not run on target boards at all, and the MN10300 and TX39 simulators can run binaries built for stdeval1 and jmr3904 target boards. The main use for a ``simulation'' configuration is if you are trying to work around problems with the device drivers or with the simulator. Also note that when using a TX39 system configured for simulator start-up you should then invoke the simulator with the `--board=jmr3904pal` option instead of `--board=jmr3904`

- If your chosen architecture does not have simulator support, then the combinations above that refer to the simulator do not apply. Similarly, if your chosen platform does not have RedBoot ROM support, the combinations listed above that use RedBoot do not apply.

The debugging environment for most developers will be either a hardware board or the simulator, in which case they will be able to select a single HAL configuration.

Configuration Tool on Windows and Linux Quick Start

This section described the GUI based configuration tool. This tool is probably more suited to users who prefer GUI's. The next section describes a CLI based tool which Unix users may prefer.

Note that the use of the Configuration Tool is described in detail in Part III, "The eCos Configuration Tool".

The Configuration Tool (see Figure 7.1, "Configuration Tool") has five main elements: the *configuration window*, the *conflicts window*, the *properties window*, the *short description window*, and the *output window*.

Figure 7.1. Configuration Tool
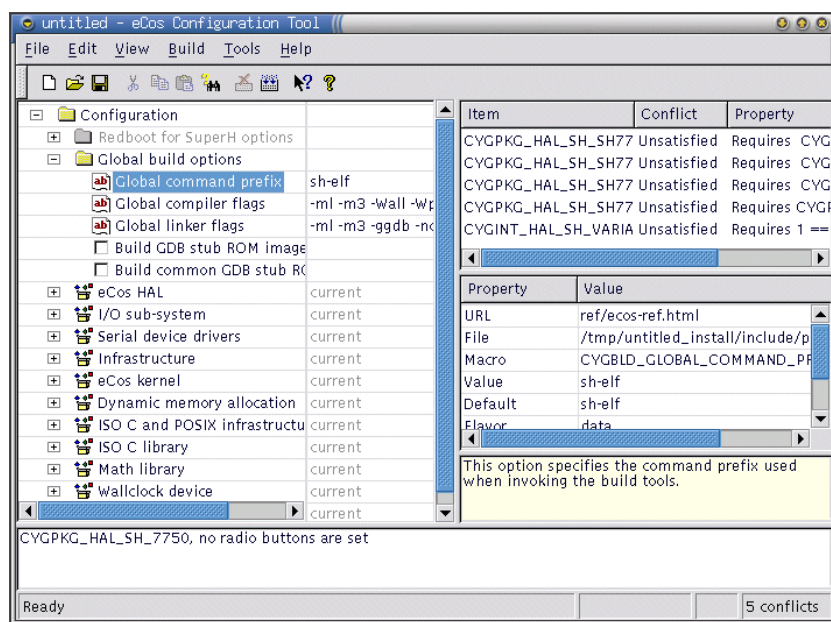
Start by creating a new configuration for the default target and template of your installation and profile from the file menu File → New (`<target>:<template>`) (**Ctrl+N**).

Figure 7.2. File Menu
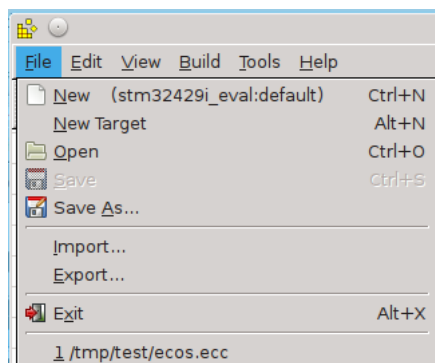
If you wish to select a different target and/or template, select from the file menu File → New Target (**Alt+N**) as illustrated in Figure 7.2, "File Menu" and the dialog illustrated in Figure 7.3, "Target and Template selection" will be displayed.

Figure 7.3. Target and Template selection
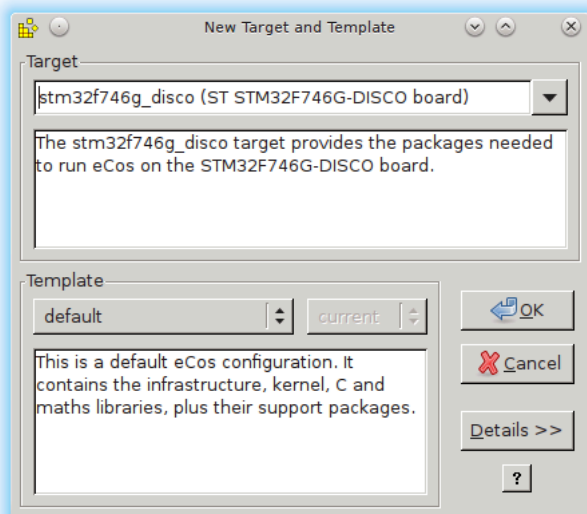
 Note

This method of switching differs from versions prior to eCosPro 4.0 where users were instructed to use the *Template Dialog* through the menu Build → Templates... (**Ctrl+M**). From version 4.0 this dialog is now only used to change the template of the current target and configuration. To switch an existing configuration to a different target see the section called "Switching Targets, Repositories and Versions".

Make sure that the configuration is correct for the target in terms of Startup type and verify endianness, CPU model, etc. (see Figure 7.4, " Configuring for the target").

Figure 7.4. Configuring for the target
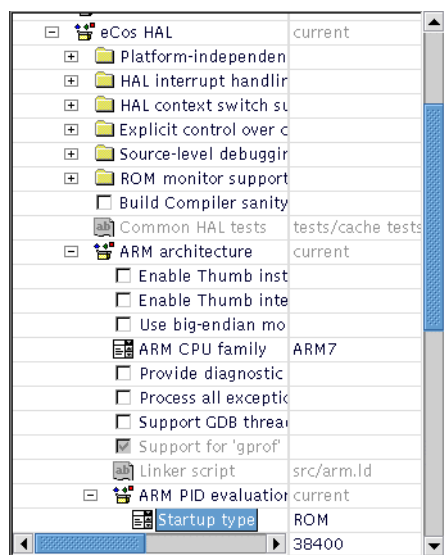
Next, select the Build → Library (**F7**) menu item to start building eCos (see Figure 7.5, "Selecting the Build Library menu item"). The application will configure the sources, prepare a build tree, and build the `libtarget.a` library, which contains the eCos kernel and other packages.

Figure 7.5. Selecting the Build Library menu item
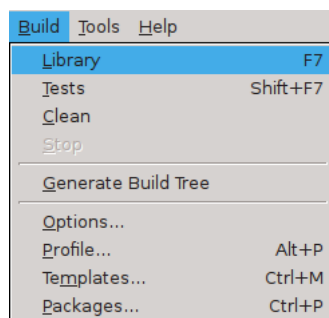
The *Save As* dialog box will appear, asking you to specify a directory in which to place your save file. You can use the default, but it is a good idea to make a subdirectory, for example `$HOME/ecos-work`.

Caution

For historic reasons, due to prior limitations on Windows you are advised not to use spaces in filenames, directories or paths.

Figure 7.6. Save file dialog
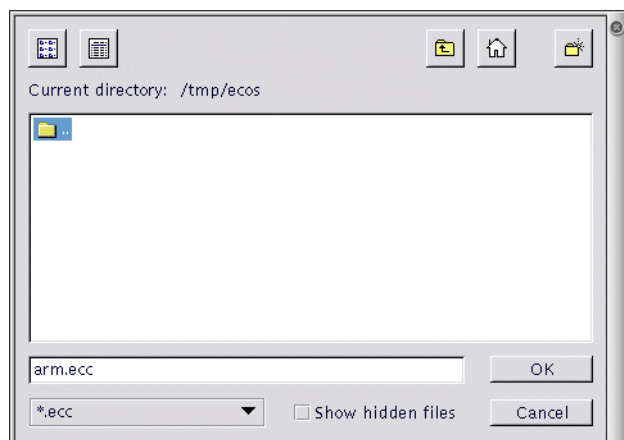
When you build an eCos library the build (**TARGET-gcc**) and host tools set by the *Profile* will be used. A set of user-defined build tools may be selected, using Tools → Paths → Build Tools, to take precedence over these using the *Build Tools* dialog box as shown in Figure 7.7, "Build tools dialog". You can select a location from the drop down list, browse to the directory using the *Browse* button, or type in the location of the build tools manually. As the build tools are architectural, this user-defined path will remain attached to the profile that was active when the path was set. When the profile is changed, this will change to the value previously associated with the new profile.

Figure 7.7. Build tools dialog
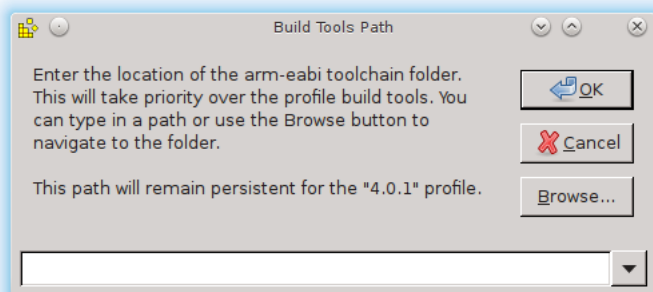
Similarly, a set of user-defined *Host Tools* may take precedence over the *Profile*'s Host Tools using Tools → Paths → Host Tools (for commands such as **make**, **cat** and **ls** on Windows hosts and **openocd** on both Linux and Windows hosts) using the *Host Tools* dialog box shown in Figure 7.8, "Host tools dialog". As with the *Build Tools* dialog, you can select a location from the drop down list, browse to the directory using the *Browse* button, or type in the location of the user tools manually. However, unlike the user-defined Build Tools, the user-defined Host tools are not associated with any *Profile* and will remain persistent between *Profile*'s and different invocations of the eCos Configuration Tool.

Figure 7.8. Host tools dialog
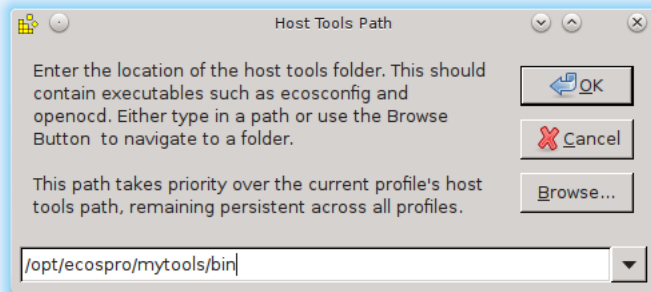
Note

Both the *Build Tools* and *User Tools* form part of an *eCosPro Profile* so it is not normally necessary on eCosPro installations to enter paths to either tools. If you do enter paths for either, these will take preference to the *Build* and *User Tools* of the selected profile. In addition, on Linux, specifying *Host Tool* will often be unnecessary as the tools will already be on your *PATH*.

When a build has been invoked, the Configuration Tool configures the sources, prepares a build tree, and builds the `libtarget.a` library into an install tree, which contains:

- the eCos kernel (`libtarget.a` library) and other libraries or packages

- the include files to be referenced by the sources of your application

- eCos configuration segments (`.ecm` files) that may be imported into the eCos configuration to enable a specific kernel, library or package behaviour

- host tool configuration files which may be used by Eclipse, the eCos Configuration Tools or developer to either run or debug both tests and your application

The build tree will be created in the same directory the saved configuration (`.ecc` file) is located and will be given the same name as the configuration file, without the `.ecc` extension, and postfixed with `_build`. Similarly, the install tree will be created in the same directory as the saved configuration and build tree but will be postfixed with `_install`.

For example, if you create a configuration file `myconfig.ecc` in the directory `C:\PlayPit`, after a successful build you will have the following configuration file and its associated build and install subdirectories in `C:\PlayPit`

- `myconfig.ecc`

- `myconfig_build`

- `myconfig_install`

The output from the configuration process and the building of `libtarget.a` will be shown in the output window.

Once the build process has finished you will have a kernel with other packages in the file `libtarget.a` located within the `lib` subdirectory of your install tree. You should now build the eCos tests for your particular configuration.

You can do this by selecting Build → Tests (**Shift-F7**). Notice that you could have selected *Tests* instead of *Library* in the earlier step and it would have built *both* the library and the tests, but this would increase the build time substantially, and if you do not need to build the tests it is unnecessary.

Figure 7.9. Selecting the Build Tests menu item
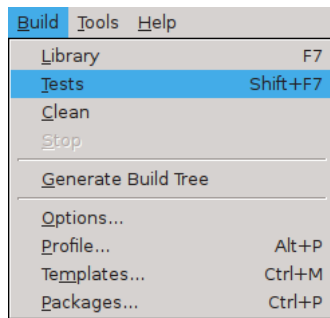
Chapter 8, *Running an eCos Test Case* will guide you through running one of the test cases you just built on the selected target, using GDB.

ecosconfig on Windows and Linux Quick Start

As an alternative to using the graphical Configuration Tool, it is possible to configure and build a kernel by editing a configuration file manually and using the **ecosconfig** command. Users with a Unix background may find this tool more suitable than the GUI tool described in the previous section.

Manual configuration and the **ecosconfig** command are described in detail in Chapter 26, *Manual Configuration*.

To use the **ecosconfig** command you need to start a shell. In Windows you need to start a DOS command line or Shell Environment. For eCosPro installations on Windows earlier than eCosPro 3.1, you need to start a CygWin **bash** shell. These may also be obtained using the menu item Tools → Shell of the GUI eCos Configuration Tool.

eCosPro installations will include Desktop menu entries for the Shell Environment for both Linux and Windows that will also set up the paths to user-defined build and user tools, as well as the tools defined by the eCosPro profiles. If you have multiple eCosPro installations or profiles, you will be prompted to select a Profile into which the Shell Environment is to be configured. This initialises the PATH environment variables and sets up the eCos specific environment variables, such as ECOS_REPOSITORY, as well as the OPENOCD_SCRIPTS, ECOS_INSTALL_DIR and ECOSPRO_PROFILE variables as described in the section called "Environment Variables".

The following instructions assume that the PATH and ECOS_REPOSITORY environment variables have been set up correctly. These instructions also assume Linux usage but can be applied equivalently to Windows.

Before invoking **ecosconfig** you need to choose a directory in which to work. For the purposes of this tutorial, the default path will be `<BASE_DIR>` (e.g. `$HOME/ecos-work`) Create this directory and change to it by typing:

```
$ mkdir <BASE_DIR>
$ cd <BASE_DIR>
```

To see what options can be used with **ecosconfig**, type:

```
$ ecosconfig --help
```

The available packages, targets and templates may be listed as follows:

```
$ ecosconfig list
```

Here is sample output from **ecosconfig** showing the usage message.

Example 7.1. Getting help from ecosconfig

```
$ ecosconfig --help --config-help
```

```
Usage: ecosconfig [ qualifier ... ] [ command ]
  commands are:
    list [ packages | templates | targets ] : list repository contents
    new TARGET [ TEMPLATE [ VERSION ] ]     : create a configuration
    target TARGET                           : change the target hardware
    template TEMPLATE [ VERSION ]           : change the template
    add PACKAGE [ PACKAGE ... ]             : add package(s)
    remove PACKAGE [ PACKAGE ... ]          : remove package(s)
    packages [ loaded | active ]            : list configuration packages
    attributes [ ATTRIBUTE ... ]            : list configuration attributes
    present PACKAGE [ PACKAGE ... ]         : check presence of package(s)
    version VERSION PACKAGE [ PACKAGE ... ] : change version of package(s)
    export FILE                             : export minimal config info
    import FILE                             : import additional config info
    check                                   : check the configuration
    resolve                                 : resolve conflicts
    tree                                    : create a build tree
    <configuration command>                 : get or set a configuration option
                                            : use --config-help for more details

  qualifiers are:
    --config=FILE                           : the configuration file
    --prefix=DIRECTORY                      : the install prefix
    --srcdir=DIRECTORY                      : the source repository
    --builddir=DIRECTORY                    : the build tree directory (default: cwd)
    --no-resolve                            : disable conflict resolution
    --version                               : show version and copyright
    -q, --quiet                             : reduce verbosity
    -v, --verbose                           : increase verbosity
    -l, --long                              : full/long list format
    -i, --ignore-errors                     : ignore unresolved conflicts
    -n, --no-updates                        : read-only mode, do not modify the file system
    -c, --compat                            : GUI configtool compatibility
    --enable-debug                          : enable debugging in this configuration
    --disable-debug                         : disable debugging in this configuration
    --help                                  : display this message
    --config-help                           : display configuration commands

Configuration commands are:
  set_comment `COMMENT'                 : set comment to COMMENT
  add_comment `COMMENT'                 : append COMMENT to comment
  get_comment                           : output the comment
  get_enabled <PACKAGE|OPTION>          : output 0 or 1 if PACKAGE or boolean OPTION is enabled
  set_enabled <PACKAGE|OPTION> <0|1>    : enable (1) or disable (0) PACKAGE or boolean OPTION
  get_value OPTION                      : output the value of cdl OPTION
  set_value OPTION VALUE                : set the value of cdl OPTION to VALUE
  get_legal_values OPTION               : output the legal values of cdl OPTION
$
```

Example 7.2. ecosconfig output — list of available packages, targets and templates

```
$ ecosconfig list
Package CYGPKG_CYGMON (CygMon support via eCos):
aliases: cygmon
versions: <version>
Package CYGPKG_DEVICES_WALLCLOCK_DALLAS_DS1742 (Wallclock driver for Dallas 1742):
aliases: devices_wallclock_ds1742 device_wallclock_ds1742
versions: <version>
Package CYGPKG_DEVICES_WALLCLOCK_SH3 (Wallclock driver for SH3 RTC module):
aliases: devices_wallclock_sh3 device_wallclock_sh3
versions: <version>
Package CYGPKG_DEVICES_WATCHDOG_ARM_AEB (Watchdog driver for ARM/AEB board):
aliases: devices_watchdog_aeb device_watchdog_aeb
versions: <version>
Package CYGPKG_DEVICES_WATCHDOG_ARM_EBSA285 (Watchdog driver for ARM/EBSA285 board):
aliases: devices_watchdog_ebsa285 device_watchdog_ebsa285
versions: <version>
…
```

Selecting a Target

To configure for a listed target, type:

```
$ ecosconfig new <target> [<template>]
```

The *<template>* is optional and if none is specified, the *default* template will be used. For example, to configure for the ST Micro 32F429IDISCOVERY development board, type:

```
$ ecosconfig new stm32f429i_disco
```

You can then edit the generated file, `ecos.ecc`, setting the options as required for the target (endianess, CPU model, Startup type, etc.). For detailed information about how to edit the `ecos.ecc` file, see the *CDL Writer's Guide* and the section called "Editing an eCos Savefile".

To create a build tree for the configured target in the current directory where `ecos.ecc` is located by type:

```
$ ecosconfig tree
```

You can use the `--builddir` to specify a directory for the build tree to be located, other than the current directory, as well as the `--config` option to specify a path to the configuration file if it is different from `ecos.ecc` or not in the current directory.

If there are any problem with the configuration, **ecosconfig** will tell you. The most likely cause of this is mistakes when editing the `ecos.ecc` file. You can check whether the configuration you have made is correct, without building the tree with the following command:

```
$ ecosconfig check
```

If this reports any conflicts you can get **ecosconfig** to try and resolve them itself by typing:

```
$ ecosconfig resolve
```

See the section called "Conflicts and constraints" for more details.

You can now run the command **make** or **make tests**, after which you will be at a similar point you would be after running the Configuration Tool. The library and header files will be in the `install` subdirectory of the current working directory (unless of course the option `--prefix` is provided to the **ecosconfig tree** command to specify the path to the install directory). You can now start developing your own applications, following the steps in Chapter 9, *Building and Running Sample Applications*.

> **Note**
>
> The build and install directory layout differ between the eCos Configuration Tool and **ecosconfig** for historic reasons. You can make **ecosconfig** use the same layout and naming convention as the eCos Configuration Tool by including the `-c` option with appropriate **ecosconfig** commands.

The procedure shown above allows you to do very coarse-grained configuration of the eCos kernel: you can select which packages to include in your kernel, and give target and start-up options. But you cannot select components within a package, or set the very fine-grained options.

To select fine-grained configuration options you will need to edit the configuration file `ecos.ecc` in the current directory and regenerate the build tree.

> **Caution**
>
> You should follow the manual configuration process described above very carefully, and you should read the comments in each file to see when one option depends on other options or packages being enabled or disabled. If you do not, you might end up with an inconsistently configured kernel which could fail to build or might execute incorrectly.

Chapter 8. Running an eCos Test Case

In the section called " Configuration Tool on Windows and Linux Quick Start" or the section called " **ecosconfig** on Windows and Linux Quick Start" you created the eCos test cases as part of the build process. Now it is time to try and run one.

Using the Configuration Tool

Test executables that have been linked using the Build→Tests (**Shift-F7**)operation against the current configuration can be executed by selecting Tools→Run Tests (**Ctrl+F5**).

This will bring up the dialog illustrated in Figure 8.1, "Run tests". For a detailed explanation details of how to configure the graphical Configuration Tool to run tests, please refer to Chapter 18, *Execution*.

Figure 8.1. Run tests
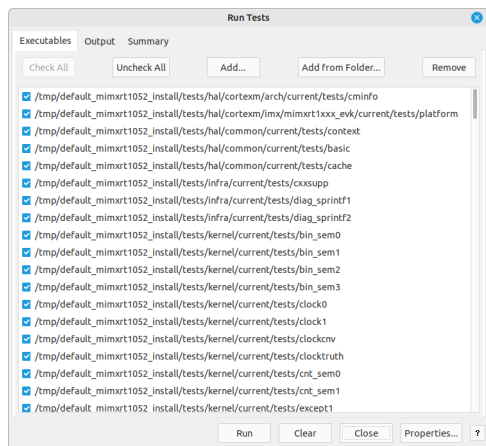
Select the *Executables* tab, press the *Uncheck All* button and then find and check just one test, for example `bin_sem0`

Now press the *Properties* button to set up communications with the target. This will bring up a properties dialog shown in Figure 8.2, "Properties dialog box".

Generally you do not need to adjust the *Download* or *Runtime* timeouts, but the *Remote* timeout may need to be set if communication between gdb and the remote target is likely to suffer from delays. For example, if the network connection to the target is over a satellite internet connection, or if you are connecting through a pipe to OpenOCD on a slow Windows host where the startup of OpenOCD can be slow, you will need to set the remote timeout to a value higher than the default of 2. Other configuration details may be found in the section called "Debug Connection".

If you have connected the target board via a serial cable, check the *Serial* radio button, and select the serial port and baud rate for the board. If the target is connected via the network or Hardware debugger with gdb server (such as the Ronetix PEED, BDI2000 or OpenOCD in server mode) select the *TCP/IP* button and enter the IP address or DNS name that the board, hardware debugger or host has been given, and the port number (usually 9000 for RedBoot and the PEEDI, 2001 for the BDI2000). If you are using a hardware debugger locally and accessing it through a pipe, select *Pipe* and enter the command to invoke the hardware debugger in pipe mode in the `Pipe Command` field. Do not include the pipe ("`|`") symbol.

Figure 8.2. Properties dialog box
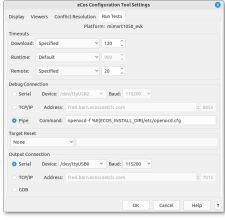
If output of the test executable is only accessible through an external source instead of the default through **GDB**, these must also be configured. See the section called "Output Connection" for further details.

Finally, if you wish to run multiple tests automatically, in *Target Reset* you may specify how the target hardware may be reset before running each test such that **GDB** can download and execute a test on the target hardware in an automated fashion. For example, if manual or physical intervention is required, such as a power cycle or pressing the reset button, set this field to *Manual*. You will then be asked by the eCos Configuration Tool to reset the board and clear a dialog Window by pressing *OK* once the target hardware has been reset and is ready to run a test. If no reset is required, for example when the board is connected to a hardware debugger that resets the board according to its configuration when a **GDB** client connects to it, set this field to *None*. If the target hardware can be reset through the local host execution of a command or script, set this field to *Command* and enter the command, complete with arguments, in the field alongside. Do not include the pipe ("|") symbol. If the command is not on the path, the full path to the command must be entered. The reset command must also terminate with a zero exit code once the target hardware has been determined to successfully reset, otherwise all following test executions are cancelled as the hardware is determined as having failed.

Click OK on this dialog and go back to the *Run Tests* dialog. Press the *Run* button and the selected test will be downloaded and run. The *Output* tab will show you how this is progressing. If it seems to stop for a long time, check that the target board is correctly connected, and that eCos has been correctly configured -- especially the start-up type.

When the program runs you should see a couple of line similar to this appear:

```
PASS:<Binary Semaphore 0 OK>
EXIT:<done>
```

This indicates that the test has run successfully.

See Chapter 18, *Execution* for further details.

Using the command line

Start a command shell (such as the Command shell in Windows) with the environment variables set as described in the toolchain documentation. Change to the directory in which you set up your build tree, and invoke GDB on the test program.

To run the **bin_sem0** test (which will test the kernel for the correct creation and destruction of binary semaphores) type:

```
$ TARGET-gdb -nw install/tests/kernel/<version>/tests/bin_sem0
```

You should see output similar to the following in the command window:

```
GNU gdb THIS-GDB-VERSION
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=THIS-HOST --target=THIS-TARGET".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

```
<http://bugzilla.ecoscentric.com/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
```

If you are trying to run a synthetic target test on Linux, skip the following connection and download steps. Otherwise, connect to the target by typing:

```
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS0
```

on Linux or

```
(gdb) set remotebaud 38400
(gdb) target remote com1
```

on Windows. To use a simulator in either host O/S use

```
(gdb) target sim
```

To use a temote host or target:

```
(gdb) target remote 192.168.2.3:9000
```

or to use a pipe command to **OpenOCD**:

```
(gdb) target remote | openocd -f <path_to_target_config_file> -c "gdb_port pipe"
```

Check the documentation for the target board for the actual baud rate to use when connecting to real targets.

You will see output similar to the following:

```
Remote debugging using ...
0x0000d50c in ?? ()
at <BASE_DIR>/kernel/<version>/src/common/kapi.cxx:345
Current language:  auto; currently c++
(gdb)
```

Or if you are using the simulator:

```
Connected to the simulator.
(gdb)
```

Now download the program to the target with

```
(gdb) load
```

You should see output similar to the following on your screen:

```
Loading section .text, size 0x4b04 lma 0x108000
Loading section .rodata, size 0x738 lma 0x10cb08
Loading section .data, size 0x1c0 lma 0x10d240
Start address 0x108000, load size 21500
Transfer rate: 24571 bits/sec, 311 bytes/write.
(gdb)
```

Now set a breakpoint to trap the target once the test has completed execution:

```
(gdb) break cyg_test_exit
(gdb)
```

If you are running the test on the target through a hardware debugger you should additionally run the following command

```
(gdb) set hwdebug on
```

This will set a flag within gdb telling it to extract diagnostics messages from a pre-determined location on reaching a pre-defined hardware breakpoint, display them on the **GDB** console, and resume execution automatically. This is useful in the absense of other external mechanisms for the target hardware to emit debug messages (e.g. through a serial port) but does also normally require enabling the eCos options `CYGFUN_HAL_DIAG_VIA_GDB_FILEIO` and `CYGFUN_HAL_GDB_FILEIO` within the eCos configuration. The **set hwdebug on** is harmless if executed when the above eCos options are not set.

You are now ready to run your program. If you type:

```
(gdb) continue
```

you will see output similar to the following:

```
Continuing.
PASS:<Binary Semaphore 0 OK>
EXIT:<done>
```

Note

If you are using a simulator or the synthetic target rather than real hardware, you must use the GDB command "run" rather than "continue" to start your program.

You can terminate your GDB session through *Control+C* if the target is still running and the breakpoint `cyg_test_exit` was not reached, otherwise it will sit in the "idle" thread and use up CPU time. This is not a problem with real targets, but may have undesirable effects in simulated or synthetic targets. Type **quit** once back at the gdb prompt and you are done.

Testing Filters

While most test cases today run solely in the target environment, some packages may require external testing infrastructure and/or feedback from the external environment to do complete testing.

The serial package is an example of this. The network package also contains some tests that require programs to be run on a host. See the network *Tests and Demonstrations* section in the network documentation in the *eCos Reference Guide*. Here we will concentrate on the serial tests since these are applicable to more targets.

Since the serial line is also used for communication with GDB, a filter is inserted in the communication pathway between GDB and the serial device which is connected to the hardware target. The filter forwards all communication between the two, but also listens for special commands embedded in the data stream from the target.

When such a command is seen, the filter stops forwarding data to GDB from the target and enters a special mode. In this mode the test case running on the target is able to control the filter, commanding it to run various tests. While these tests run, GDB is isolated from the target.

As the test completes (or if the filter detects a target crash) the communication path between GDB and the hardware target is re-established, allowing GDB to resume control.

In theory, it is possible to extend the filter to provide a generic framework for other target-external testing components, thus decoupling the testing infrastructure from the (possibly limited) communication means provided by the target (serial, JTAG, Ethernet, etc).

Another advantage is that the host tools do not need to know about the various testing environments required by the eCos packages, since all contact with the target continues to happen via GDB.

Chapter 9. Building and Running Sample Applications

The example programs in this tutorial are included, along with a `Makefile`, in the `examples` directory of the eCos distribution. The first program you will run is a *hello world*-style application, then you will run a more complex application that demonstrates the creation of threads and the use of cyg_thread_delay(), and finally you will run one that uses clocks and alarm handlers.

The `Makefile` depends on an externally defined variable to find the eCos library and header files. This variable is `ECOS_INS-TALL_DIR` and must be set to the pathname of the install directory created in the section called " Configuration Tool on Windows and Linux Quick Start".

> **Notes**
>
> • The variable `ECOS_INSTALL_DIR` is set in the environment of any shell started by the current eCos Configuration Tool and need not be specified from such shells. See the section called "Environment Variables".
>
> • The `Makefile` of older versions of eCos and eCosPro used the variable `INSTALL_DIR`, did not make certain sanity checks and did not provide any useful error messages. `INSTALL_DIR` may still be used although if `ECOS_INSTALL_DIR` is set in the environment, or passed as an override to **make**, it will take precedence.

`ECOS_INSTALL_DIR` may be either be set in the shell environment or may be supplied on the command line. It is set by within shells opened by the eCos Configuration Tool using Tools → Shell. To set it in another shell do the following in a **bash** shell where `<BASE_DIR>` is the base directory containing the `_install` directory (e.g. `$HOME/ecos-work`):

```
$ export ECOS_INSTALL_DIR=<BASE_DIR>/arm_install
```

You can then run **make** without any extra parameters to build the examples.

Alternatively, if you can do the following:

```
$ make ECOS_INSTALL_DIR=<BASE_DIR>/arm_install
```

eCos Hello World

The following code is found in the file `hello.c` in the `examples` directory:

eCos hello world program listing

```
/* this is a simple hello world program */
  #include <stdio.h>
  int main(void)
  {
  printf("Hello, eCos world!\n");
  return 0;
}
```

To compile this or any other program that is not part of the eCos distribution, you can follow the procedures described below. Type this explicit compilation command (assuming your current working directory is also where you built the eCos kernel):

```
$ TARGET-gcc -g -I<BASE_DIR>/install/include hello.c -L<BASE_DIR>/install/lib -Ttarget.ld -nostdlib
```

The compilation command above contains some standard GCC options (for example, `-g` enables debugging), as well as some mention of paths (`-I<BASE_DIR>/install/include` allows files like `cyg/kernel/kapi.h` to be found, and `-L<BASE_DIR>/install/lib` allows the linker to find `-Ttarget.ld`).

The executable program will be called `a.out`.

Note

Some target systems require special options to be passed to gcc to compile correctly for that system. Please examine the Makefile in the examples directory to see if this applies to your target.

You can now run the resulting program using GDB in exactly the same the way you ran the test case before. The procedure will be the same, but this time run ***TARGET*-gdb** specifying `-nw a.out` on the command line:

```
$ TARGET-gdb -nw a.out
```

For targets other than the synthetic linux target, you should now run the usual GDB commands described earlier. Once this is done, typing the command "continue" at the (gdb) prompt ("run" for simulators) will allow the program to execute and print the string "Hello, eCos world!" on your screen.

On the synthetic linux target, you may use the "run" command immediately - you do not need to connect to the target, nor use the "load" command.

A Sample Program with Two Threads

Below is a program that uses some of eCos' system calls. It creates two threads, each of which goes into an infinite loop in which it sleeps for a while (using cyg_thread_delay()). This code is found in the file `twothreads.c` in the examples directory.

eCos two-threaded program listing

```
#include <cyg/kernel/kapi.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* now declare (and allocate space for) some kernel objects,
  like the two threads we will use */
cyg_thread thread_s[2]; /* space for two thread objects */

char stack[2][4096]; /* space for two 4K stacks */

/* now the handles for the threads */
cyg_handle_t simple_threadA, simple_threadB;

/* and now variables for the procedure which is the thread */
cyg_thread_entry_t simple_program;

/* and now a mutex to protect calls to the C library */
cyg_mutex_t cliblock;

/* we install our own startup routine which sets up threads */
void cyg_user_start(void)
{
  printf("Entering twothreads' cyg_user_start() function\n");

  cyg_mutex_init(&cliblock);

  cyg_thread_create(4, simple_program, (cyg_addrword_t) 0,
  "Thread A", (void *) stack[0], 4096,
  &simple_threadA, &thread_s[0]);
  cyg_thread_create(4, simple_program, (cyg_addrword_t) 1,
  "Thread B", (void *) stack[1], 4096,
  &simple_threadB, &thread_s[1]);
```

```
  cyg_thread_resume(simple_threadA);
  cyg_thread_resume(simple_threadB);
}

/* this is a simple program which runs in a thread */
void simple_program(cyg_addrword_t data)
{
  int message = (int) data;
  int delay;

  printf("Beginning execution; thread data is %d\n", message);

  cyg_thread_delay(200);

  for (;;) {
    delay = 200 + (rand() % 50);

    /* note: printf() must be protected by a call to cyg_mutex_lock() */
    cyg_mutex_lock(&cliblock);
    printf("Thread %d: and now a delay of %d clock ticks\n", message, delay);
    cyg_mutex_unlock(&cliblock);
    cyg_thread_delay(delay);
  }
}
```

When you run the program (by typing **continue** at the (*gdb*) prompt) the output should look like this:

```
Starting program: <BASE_DIR>/examples/twothreads.exe
Entering twothreads' cyg_user_start()
function
Beginning execution; thread data is 0
Beginning execution; thread data is 1
Thread 0: and now a delay of 240 clock ticks
Thread 1: and now a delay of 225 clock ticks
Thread 1: and now a delay of 234 clock ticks
Thread 0: and now a delay of 231 clock ticks
Thread 1: and now a delay of 224 clock ticks
Thread 0: and now a delay of 249 clock ticks
Thread 1: and now a delay of 202 clock ticks
Thread 0: and now a delay of 235 clock ticks
```

Note

When running in a simulator the delays might be quite long. On a hardware board (where the clock speed is 100 ticks/second) the delays should average to about 2.25 seconds. In simulation, the delay will depend on the speed of the host processor and will almost always be much slower than the actual board. You might want to reduce the delay parameter when running in simulation.

Figure 9.1, "Two threads with simple print statements after random delays" shows how this multitasking program executes. Note that apart from the thread creation system calls, this program also creates and uses a *mutex* for synchronization between the printf() calls in the two threads. This is because the C library standard I/O (by default) is configured not to be thread-safe, which means that if more than one thread is using standard I/O they might corrupt each other. This is fixed by a mutual exclusion (or *mutex*) lockout mechanism: the threads do not call printf() until cyg_mutex_lock() has returned, which only happens when the other thread calls cyg_mutex_unlock().

You could avoid using the mutex by configuring the C library to be thread-safe (by selecting the component CYGSEM_LIBC_ST-DIO_THREAD_SAFE_STREAMS).

Figure 9.1. Two threads with simple print statements after random delays
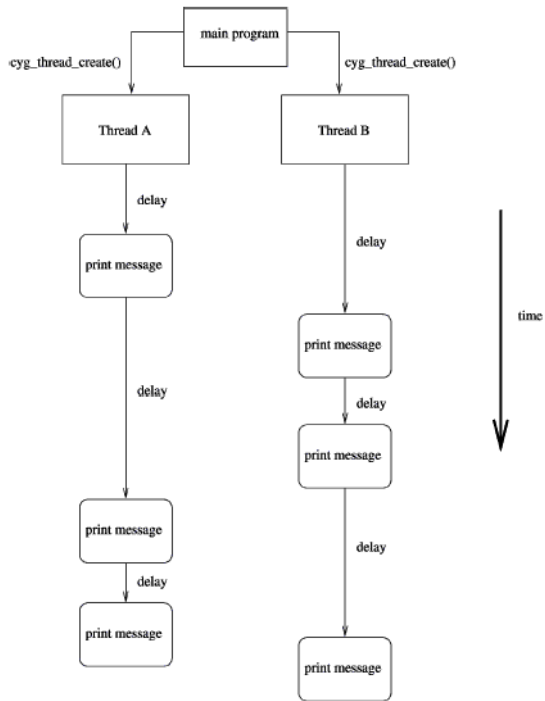
Chapter 10. More Features — Clocks and Alarm Handlers

If a program wanted to execute a task at a given time, or periodically, it could do it in an inefficient way by sitting in a loop and checking the real-time clock to see if the proper amount of time has elapsed. But operating systems usually provide system calls which allow the program to be informed at the desired time.

eCos provides a rich timekeeping formalism, involving *counters*, *clocks*, *alarms*, and *timers*. The precise definition, relationship, and motivation of these features is beyond the scope of this tutorial, but these examples illustrate how to set up basic periodic tasks.

Alarms are events that happen at a given time, either once or periodically. A thread associates an alarm handling function with the alarm, so that the function will be invoked every time the alarm "goes off".

A Sample Program with Alarms

`simple-alarm.c` (in the examples directory) is a short program that creates a thread that creates an alarm. The alarm is handled by the function `test_alarm_func()`, which sets a global variable. When the main thread of execution sees that the variable has changed, it prints a message.

Example 10.1. A sample program that creates an alarm

```
/* this is a very simple program meant to demonstrate
 a basic use of time, alarms and alarm-handling functions  in eCos */

#include <cyg/kernel/kapi.h>

#include <stdio.h>

#define NTHREADS 1
#define STACKSIZE 4096

static cyg_handle_t thread[NTHREADS];

static cyg_thread thread_obj[NTHREADS];
static char stack[NTHREADS][STACKSIZE];

static void alarm_prog( cyg_addrword_t data );

/* we install our own startup routine which sets up
  threads and starts the scheduler */
void cyg_user_start(void)
{
  cyg_thread_create(4, alarm_prog, (cyg_addrword_t) 0,
    "alarm_thread", (void *) stack[0],
    STACKSIZE, &thread[0], &thread_obj[0]);
  cyg_thread_resume(thread[0]);
}

/* we need to declare the alarm handling function (which is
 defined below), so that we can pass it to  cyg_alarm_initialize() */
cyg_alarm_t test_alarm_func;

/* alarm_prog() is a thread which sets up an alarm which is then
 handled by test_alarm_func() */
static void alarm_prog(cyg_addrword_t data)
{
  cyg_handle_t test_counterH, system_clockH, test_alarmH;
  cyg_tick_count_t ticks;
  cyg_alarm test_alarm;
```

```
  unsigned how_many_alarms = 0, prev_alarms = 0, tmp_how_many;

  system_clockH = cyg_real_time_clock();
  cyg_clock_to_counter(system_clockH, &test_counterH);
  cyg_alarm_create(test_counterH, test_alarm_func,
    (cyg_addrword_t) &how_many_alarms,
    &test_alarmH, &test_alarm);
  cyg_alarm_initialize(test_alarmH, cyg_current_time()+200, 200);

  /* get in a loop in which we read the current time and
    print it out, just to have something scrolling by */
  for (;;) {
    ticks = cyg_current_time();
    printf("Time is %llu\n", ticks);
    /* note that we must lock access to how_many_alarms, since the
       alarm handler might change it. this involves using the
       annoying temporary variable tmp_how_many so that I can keep the
       critical region short */
    cyg_scheduler_lock();
    tmp_how_many = how_many_alarms;
    cyg_scheduler_unlock();
    if (prev_alarms != tmp_how_many) {
      printf(" --- alarm calls so far: %u\n", tmp_how_many);
      prev_alarms = tmp_how_many;
    }
    cyg_thread_delay(30);
  }
}

/* test_alarm_func() is invoked as an alarm handler, so
   it should be quick and simple. in this case it increments
   the data that is passed to it. */
void test_alarm_func(cyg_handle_t alarmH, cyg_addrword_t data)
{
  ++*((unsigned *) data);
}
```

When you run this program (by typing **continue** at the (*gdb*) prompt) the output should look like this:

```
Starting program: <BASE_DIR>/examples/simple-alarm.exe
Time is 0
Time is 30
Time is 60
Time is 90
Time is 120
Time is 150
Time is 180
Time is 210
--- alarm calls so far: 1
Time is 240
Time is 270
Time is 300
Time is 330
Time is 360
Time is 390
Time is 420
--- alarm calls so far: 2
Time is 450
Time is 480
```

Note

When running in a simulator the delays might be quite long. On a hardware board (where the clock speed is 100 ticks/second) the delays should average to about 0.3 seconds (and 2 seconds between alarms). In simulation, the delay will depend on the speed of the host processor and will almost always be much slower than the actual board. You might want to reduce the delay parameter when running in simulation.

Here are a few things you might notice about this program:

- It used the `cyg_real_time_clock()` function; this always returns a handle to the default system real-time clock.

- Clocks are based on counters, so the function `cyg_alarm_create()` uses a counter handle. The program used the function `cyg_clock_to_counter()` to strip the clock handle to the underlying counter handle.

- Once the alarm is created it is initialized with `cyg_alarm_initialize()`, which sets the time at which the alarm should go off, as well as the period for repeating alarms. It is set to go off at the current time and then to repeat every 200 ticks.

- The alarm handler function `test_alarm_func()` conforms to the guidelines for writing alarm handlers and other delayed service routines: it does not invoke any functions which might lock the scheduler. This is discussed in detail in the *eCos Reference Manual*, in the chapter *The eCos Kernel*.

- There is a *critical region* in this program: the variable `how_many_alarms` is accessed in the main thread of control and is also modified in the alarm handler. To prevent a possible (though unlikely) race condition on this variable, access to `how_many_alarms` in the principal thread is protected by calls to `cyg_scheduler_lock()` and `cyg_scheduler_unlock()`. When the scheduler is locked, the alarm handler will not be invoked, so the problem is averted.

Part III. The eCos Configuration Tool

Table of Contents

Chapter 11. Getting Started

Introduction

The eCos Configuration Tool is used to tailor eCos at source level, prior to compilation or assembly, and provides a configuration file and a set of files used to build user applications. The sources and other files used for building a configuration are provided in a *component repository*, which is loaded when the eCos Configuration Tool is invoked. The component repository includes a set of files defining the structure of relationships between the Configuration Tool and other components, and is written in a *Component Definition Language* (CDL). For a description of the concepts underlying component configuration, see Chapter 22, *CDL Concepts*.

> **Note**
>
> This documentation relates to version 4.0 of the eCosPro eCos Host Tools and above. If you are using an earlier version of the eCosPro eCos Host Tools, the behaviour and screenshots documented may be slightly different from what you could encounter and certain features may not be available to you.

Environment Variables

The eCos and eCosPro host tools make use of shell environment variables to specify either locations of directory structures and files, or settings in use by the tools. These variables are:

ECOS_REPOSITORY

The `ECOS_REPOSITORY` environment variable must point to the `packages` subdirectory of the eCosPro installation (e.g. `/opt/ecospro/ecos-4.0.1/packages` on Linux or `C:\eCosPro\ecos-4.0.1\packages` on Windows).

`ECOS_REPOSITORY` may also point to multiple eCos component repositories, in which case it contains a path to each repository's `packages` subdirectory seperated by the `PATH` seperator of the host O/S. (i.e. a colon on Linux and a semi-colon on Windows).

ECOS_INSTALL_DIR

The `ECOS_INSTALL_DIR` environment variable is set to the `install` subdirectory of a prepared eCos build tree created by the eCos Configuration Tool. It will be set within any shell created using Tools → Shell as well prior to any build or test execution by the eCos Configuration Tool.

For example, this variable is referenced by the default pipe command of the eCos Configuration Tool using the `%E(ECOS_INSTALL_DIR)` macro to locate the default target-specific **openocd** script placed at `install/etc/openocd.cfg` (if supported) from the build of an eCos configuration.

OPENOCD_SCRIPTS

The `OPENOCD_SCRIPTS` environment variable points to the shared directory location where the common scripts for **openocd** may be found. These scripts are referenced by the target-specific **openocd** configuration files and may vary if different versions of the eCosPro host tools (which includes **openocd**), or a different source of **openocd**, have been installed. Normally for a single **openocd** installation this will be `/opt/ecospro/ecoshosttools/share/openocd/scripts` on Linux or `C:\eCosPro\ecoshosttools\share\openocd\scripts` on Windows.

ECOS_TARGET

This environment variable specifies the primary name of the target of the eCos Configuration Tool's configuration when the shell was created. It may be used within automated build scripts as seen fit by the developer.

ECOSPRO_PROFILE

This environment variable specifies the name of the currently active profile when the shell was created. It may be used within automated build scripts as seen fit by the developer.

PATH   This system environment variable specifies the list of directories, in order, to be searched for executables that may be executed directly from the command line without specifying the full path. If the user has specified either *Build Tools* or *Host Tools* paths, these are prepended by the eCos Configuration Tool to the *PATH* along with the GNU Toolchain and Host Tools executables directories as specified by the eCosPro Profile whenever a build is invoked, test executed or shell opened.

Note

These environment variables are persistent in a shell created by the eCos Configuration Tool and will represent the active profile and last saved configuration at the time the shell was spawned. If the profile is changed, or the name/location of the configuration file is changed, within the eCos Configuration Tool, these values will not be updated within the shell.

The chapter Environment Variables in the eCosPro Developer's Kit - Installation and Getting Started Guide provide an example of typical use.

Invoking the eCos Configuration Tool

On Linux

Invoke the configuration tool using one of the following methods:

- Directly from a command line by specifying the full path to it from the command line: The exact location is dependant on where it was installed and by whom. For example:

```
> /opt/ecos/ecoshosttools/bin/configtool        # Global installation
> $HOME/ecospro/ecoshosttools/bin/configtool    # Local installation
```

- Add the executable install directory for the eCos Configuration Tool to your PATH, and run **configtool** from your command line. For example:

```
> export PATH=/opt/ecos/ecoshosttools/bin:$PATH
> configtool
```

- Invoking it from the Desktop as illustrated in Figure 11.1, " Linux Menu ".

Figure 11.1. Linux Menu
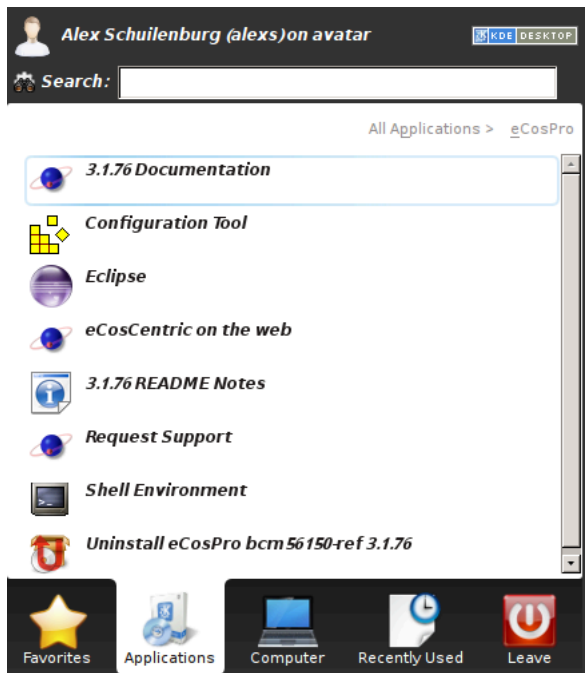
The exact representation is dependent on your Linux desktop environment. The eCosPro installer uses the FreeDesktop.org's **xdg-desktop-menu** command, if available, to install a desktop menu item for the eCos configuration tool.

- Run the application from the *Run Application...* dialog.

- Invoking it from your File Manager

You may run configtool with zero, one or two arguments. You can specify the eCos repository location, and/or an eCos save file (extension .ecc) on the command line. The ordering of these two arguments is not significant. For example:

```
configtool /opt/ecos/ecos<version>/packages myfile.ecc
```

Once started, the Configuration Tool will be displayed (see the section called "Configuration Tool").

On Windows

There are four ways in which to invoke the eCos Configuration Tool:

- from the desktop explorer or program set up at installation time. By default on Windows 7 and below this is: Start → Programs → eCosPro → Configuration Tool. On Windows 10 this is: [⊞] → All Apps → eCosPro → Configuration Tool.

- navigate to the executable directory where **ConfigTool.exe** was installed using Windows Explorer and double-click on the command.

- type (at a command prompt or in the *Start* menu,s *Run* item): ***<foldername>*\ConfigTool.exe** where ***<foldername>*** is the full path of the directory in which you installed the eCos Configuration Tool.

- open an eCos Command shell (by default on Windows 7: Start → Programs → eCosPro → Command Shell) and run the command **ConfigTool**.

Figure 11.2. Windows 10 Menu
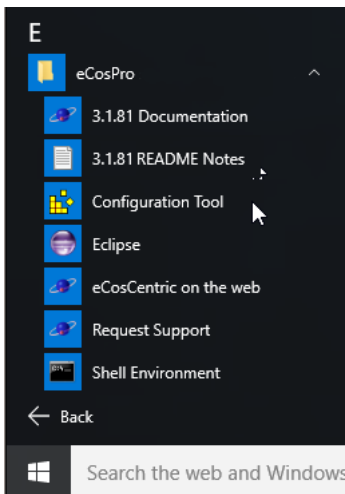
The Windows 10 Menu items added on installation are illustrated in Figure 11.2, " Windows 10 Menu ".

Once started, the Configuration Tool will be displayed (see the section called "Configuration Tool").

Configuration Tool

If multiple eCosPro installations exist you will normally be prompted to select the profile of an installation before the main configuration window is displayed. See Figure 11.4, "Profile selection dialog" in the section called "Profiles".

Figure 11.3. Configuration Tool
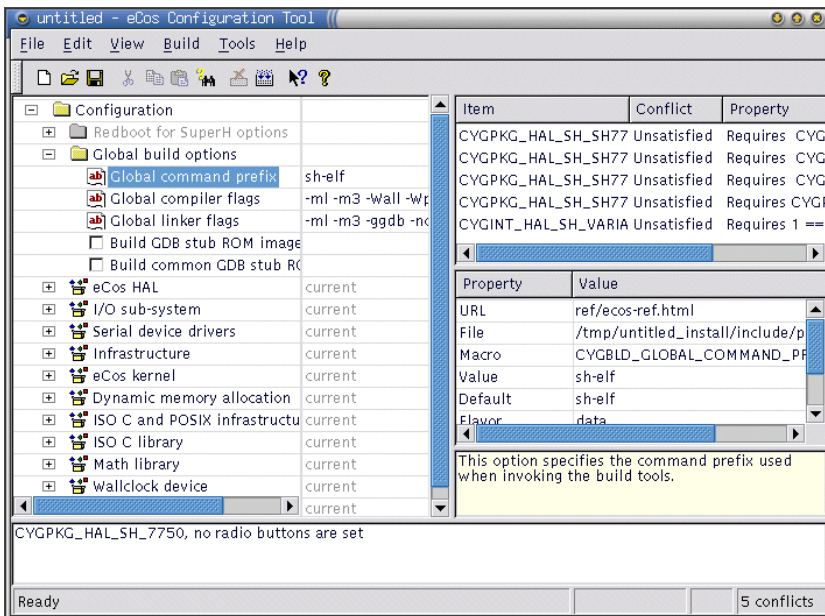
Once started, the Configuration Tool main window is displayed as illustrated in Figure 11.3, " Configuration Tool ".

You may run configtool with zero, one or two arguments. You can specify the eCos repository location, and/or an eCos save file (extension .ecc) on the command line. The ordering of these two arguments is not significant. For example:

```
configtool "c:\eCosPro\ecos-3.1.79\packages" myfile.ecc
```

If a profile name matching the profile used to create the eCos exists on your installation, it will automatically be selected and the user will not be prompted to select a profile as described in the section called "Profiles".

If you invoke the configuration tool from the command line with *--help*, you will see this output:

```
Usage: configtool [-ceEhpPsvV] [-p <str>] [input file 1] [input file 2]
  -c, --compile-help  compile online help only
  -e, --edit-only     edit save file only
  -E, --eclipse       Eclipse mode to disable certain options
  -h, --help          displays help on the command line parameters
  -p, --profile=<str> specifies the eCosPro profile to use
  -P, --edit-profile  create, edit or delete profiles only
  -s, --shell         Open shell with selected profile
  -v, --version       print version
  -V, --Verbose       be verbose
```

This summarizes valid parameters and switches. Switches are shown with both short form and long form.

--compile-help compiles help contents files from the HTML documentation files that the tool finds in the eCos repository, and exits.

--edit-only runs the Configuration Tool in a mode that suppresses creation of a build tree, in case you only want to create and edit save files.

--eclipse runs the Configuration Tool in a mode that is used by the eCosCentric eCosPro Eclipse plug-in. This mode restricts the configuration tool to a specified eCos Configuration file and profile by disabling certain menu options, such as File → Open (**Ctrl +O**), File → Save As..., File → History, Build → Profile (**Alt+P**) and the modification of the Build (Toolchain) Tools, Host Tools and Paths. This allows eCosPro Eclipse plugin users to operate only within the corresponding Eclipse project.

--help shows valid options and parameters, as above.

--profile=<profile> selects the profile `<profile>` on startup, bypassing the profile dialog selection. The legacy profile may be provided as either `env:ECOS_REPOSITORY` or as the escaped environment variable name (i.e. so it is not expanded to the environment variable value from the command line).

--edit-profile opens the profile dialog to allow editing, adding or deletion of profiles without invoking the configuration tool. This feature is particlarly useful when you only have one profile, which is also the active profile, and wish to edit it.

--shell opens the profile dialog to allow selection of a profile and opens a development shell (command line shell) for the selected profile. This is equivalent to opening the configuration tool, selecting the menu options *Tools>Shell* and exiting the configuration tool.

--version shows version and build date information, and exits.

--Verbose be verbose on the command line when running.

Profiles and the Component Repository

When you invoke the eCos Configuration Tool, it accesses the Component Repository, a read-only location of configuration information. For an explanation of "Component Repository" see Chapter 22, *CDL Concepts*.

Profiles

eCosPro 3.1 introduced support for *repository profiles* (*profiles*) which make it easier to have more than one component repository (i.e. eCos repository or eCosPro installation) present at the same time. Each eCosPro installation defines its own *system profile* which declares what it is, where its component repository is located, what toolchain it requires and its default hardware target and template. Version 4.0 of the eCosPro Configuration Tool introduced *user profiles* to allow users to define their own "Component Repository" and toolchain set.

> **Note**
>
> Editing of *system profiles* and the active profile is not permitted.

The Configuration Tool looks in two places for profile definitions, a *system-wide* file and a *per-user* file. Any definitions present in the per-user file override those in the system-wide file.

Table 11.1. Profile definition file locations

Platform	System-wide	Per-user
Windows	`%ALLUSERSPROFILE%` `\ecospro.ini`	`%LOCALAPPDATA%\ecospro.ini` or `%APPDATA%\ecospro.ini`
Linux	`/etc/ecospro.conf`	`$HOME/.ecospro.conf`

At startup, the Configuration Tool reads the profile definitions and checks them for validity. If only one valid profile is found, or a profile is specified on startup using the *--profile=<profile>* command line option, or an eCosPro configuration file is specified on startup which contains information regarding the profile against which that configuration was created, that *profile* is selected unconditionally. If none of the above conditions are satisfied, you will be prompted to select or create a profile to use.

To select a profile, highlight the descriptive name of the profile, the details of which will then appear in the *Preview* area, and then select OK.

Once selected, the descriptive name of the currently active profile is displayed in the status bar at the bottom right of the Configuration Tool window.

Figure 11.4. Profile selection dialog
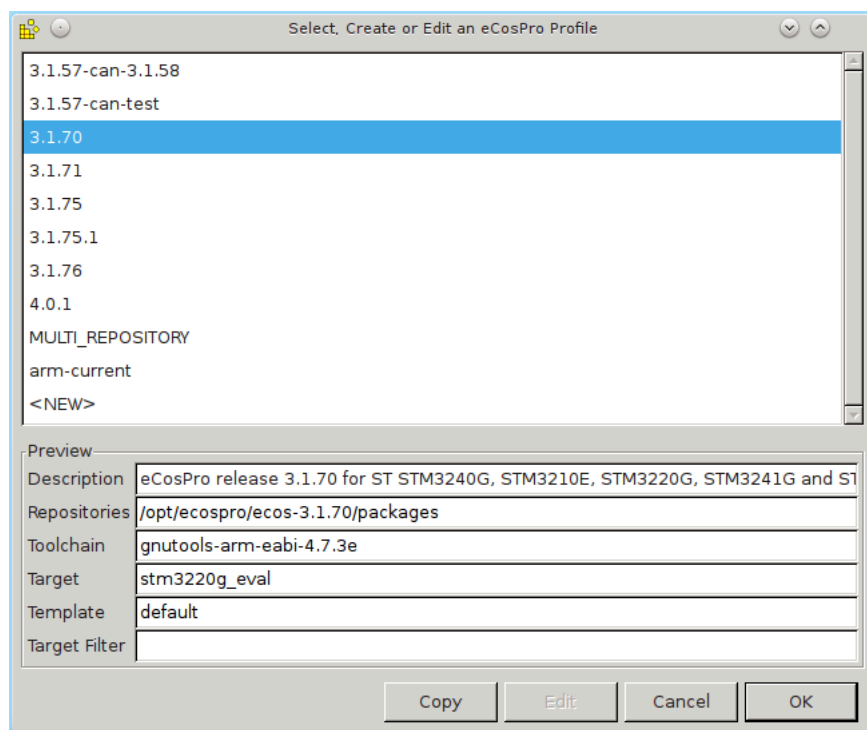
> **Note**
>
> You cannot *Edit* the current active profile.

Legacy Profile and Compatibility with eCosPro v2.0.x and 3.0.x

Older installations of eCosPro created `ecosenv.sh` and/or `ecosenv-<version>.sh` files which set the `ECOS_REPOSITORY` environment variable for you.

In order to co-exist with such installations, this version of the Configuration Tool also examines the value of `ECOS_REPOSITORY` when it starts up: if it is set and **not equal** to any repository in the available profiles, a legacy profile is created. The legacy profile is a system profile that cannot be edited and will appear in the profile selection dialog as either `%ECOS_REPOSITORY%` for Windows Hosts or `${ECOS_REPOSITORY}` for Linux Hosts.

Alternatively, if an eCos repository is passed on the command line to the configuration tool, either as a directory or the value of an environment variable, the legacy profile as described above will be added using the repository provided as the eCos repository. In this case the environment variable `ECOS_REPOSITORY` is ignored when creating the legacy profile.

> **Note**
>
> When using this backwards compatibility mode, you may need to set up the path to the appropriate toolchain for your target hardware using the Tools → Paths → Build Tools menu item. Alternatively, you can ensure the toolchain is on the `PATH` before starting the Configuration Tool. Older releases will have used the files `ecosenv.sh` and/or `ecosenv-<version>.sh` in order to configure appropriate settings.

Migrating configurations to a new repository profile

It is possible to migrate a configuration from one profile to another which provides the same set of packages (for example, when upgrading to another release of eCosPro which targets the same hardware platform).

To do this, first open the configuration in its current profile, then use the Build → Profile (**Alt+P**) menu item to select a different profile. You will be prompted to save the configuration, if necessary, before selecting the new profile; the Configuration Tool then attempts to migrate the configuration to the new profile.

The migration process will result in a number of warnings from CDL, mostly to the effect that the previously-selected versions of eCos packages were not found and that the available versions were used instead. There may also be conflicts arising, which you will be prompted to resolve in the usual way.

If the configuration could not be automatically migrated, the action is aborted and you are warned that that save file was "invalid" for that profile. This normally indicates that a package loaded by the existing configuration was not present in the new repository; details will appear in the error output.

You can check the location of the current repository, the current save file path, the current hardware template, package deplate and any packages added or removed from your configuration by selecting Help → Repository Information (**Ctrl+I**). A summary will be displayed along with any licensing information (this feature was added in eCosPro 4.0).

Profile Dialog

The creation and editing of user profiles is done through the *Profile Dialog* selection window (see Figure 11.4, "Profile selection dialog"). This dialog may be reached either on startup of the configuration tool when prompted to select a profile, or through the profile selection menu option Build → Profile (**Alt+P**).

Creating or editing user profiles

To create a new profile, within the profile selection dialog select the `<NEW>` profile from the list of profiles and press `EDIT`. Similarly, to edit or copy an existing profile, select the profile from the list and select either `EDIT` or `COPY` respectively.

Note

- Only user profiles that are not also the active profile (used by the current configuration) may be edited.

- Invoking the **configtool** with the argument `--edit-profiles` will allow you to edit all *user profiles* and exit the eCos Configuration Tool immediately after editing.

Figure 11.5. Profile editing dialog
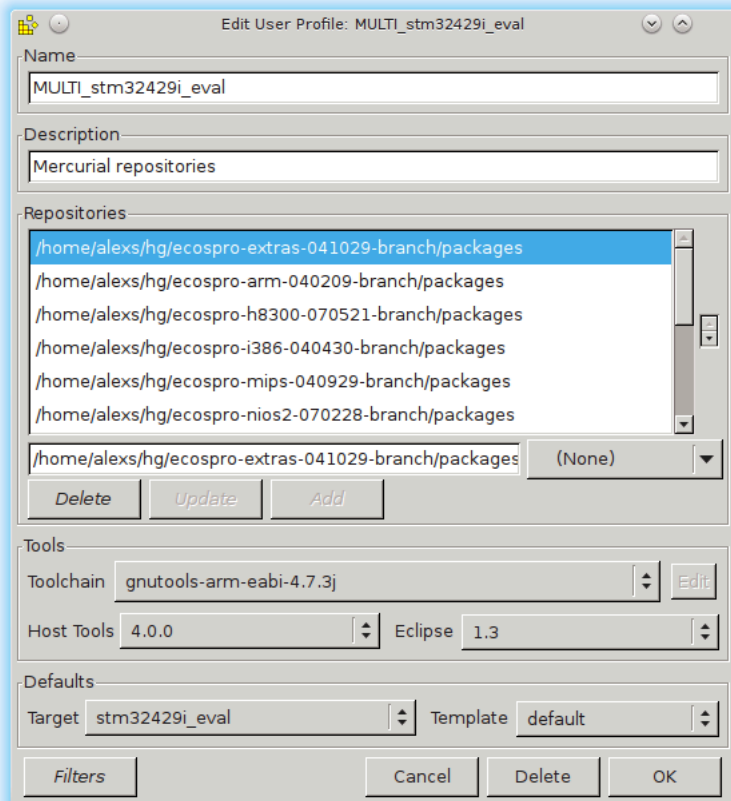
This will open the new dialog window illustrated above. The fields are as follows:

Name
: The descriptive name that appears at the bottom right of the Configuration Tool and in the Profile selection dialog.

Description
: A textual description of the profile to appear in the preview field of the Profile selection dialog.

Repositories
: The list of directories containing eCos repositories, in the order in which they are to be used. For normal eCosPro installations this will contain a single directory. Multiple directories may be specified from which packages can be selected when creating an eCos configuration. If the same package, as determined by the macro name, appears in multiple directories, the package within the highest directory repository of the list will be used.

: The order of the repositories may be altered by selecting a directory from the list and using the up/down arrows to the right of the list to move the directory up or down in the list order.

The order of the repositories may be altered by selecting a directory from the list and using the up/down arrows to the right of the list to move the directory up or down in the list order.

Adding a repository

A new repository may be added by entering the path to the `packages` directory containing the `ecos.db` file in the input field below the list of repositories, or by choosing the directory using the selector to the right of this input field.

The `ADD` button will become enabled once a valid directory has been entered and must be pressed to add the respository directory to the list.

Deleting a repository

To delete a directory from the list of repositories, select the directory in the list and press the `DELETE` button.

Changing a repository

To alter a directory in the list of repositories, select the directory in the list, alter the name of the directory in the field above or select the new `packages` directory and press the `UPDATE` button. This button will only become enabled when the direcory field contains a valid path to a directory containing an `ecos.db` file.

Toolchain

eCosPro profiles include the GNU toolchain to be used to build an eCos configuration. GNU toolchains (compiler, debugger and utilities) are provided with eCosPro Developers Kits which may be periodically updated. *User profiles* allow the developer to select a newer or different GNU toolchain, including a GNU toolchain from a different source, for use when building an eCos configuration.

The *Toolchain* drop-down list within the *Tools* section of the dialog allows the user to specify the descriptive name of the toolchain for the profile. The path to the executables (**gcc**, **g++**, and family) for this toolchain is added to the front of the user's search path to these executables are used when the eCos configuration (and optional tests) is built.

The `%PATH%` toolchain on Windows or `${PATH}` on Linux allow the user to use the first GNU toolchain executables found in the user's search path.

A new toolchain may be specified by selecting the `<NEW>` followed by pressing the `<EDIT>` button which will cause a new dialog to be opened that allows the user to specify a new toolchain. Similarly, existing user-specified toolchains may be edited by selecting the toolchain from the drop-down list and pressing the `<EDIT>` button. See the section called "Creating or editing user-defined toolchains" for more details.

Host Tools

Included in eCosPro profiles are the eCosCentric provided Host Tools and Eclipse are to be used within a *Profile*. The *Host Tools* drop-down list within the *Tools* section of the dialog allows the user to specify the descriptive name of the Host Tools installation for the profile. The path to the executables (**openocd**, **ecosconfig**, **mk_romfs instdump** etc. and, on Windows, additional tools such as **bash**, **make**, **patch**) is added to the front of the user's search path when the eCos configuration (and optional tests) is built, by Eclipse when building the eCos library or application, and when a new shell is opened by the configuration tool.

The `%PATH%` for Host Tools on Windows or `${PATH}` on Linux will not assign any Host Tools to the *Profile* and will user's the seach path.

> ### Note
>
> There is no means within the eCos Configuration Tool to add a new *Host Tool* definition to the drop down.

Eclipse	Similar to *Host Tools*, the *Eclipse* drop-down list within the *Tools* section of the dialog allows the user to specify the descriptive name of Eclipse installation for the profile.
	The `%PATH%` for Eclipse on Windows or `${PATH}` on Linux will not assign any Eclipse installation to the *Profile*.

> **Note**
>
> This item is not currently used by anything but may be used in future by the eCos Configuration Tool.

Target and Template defaults	When creating a new eCos configuration from the menu using File → New (**Ctrl+N**) a default `<target>` and `<template>` are used. These defaults can be set through the corresponding drop-down selections within the *Defaults* section of the dialog.
Filters	eCosPro 4.0 introduced *CDL attributes* to eCos packages and targets as an aid for developers to search for or view packages and targets with specific attributes.
	One use of this feature is to restrict the number of targets listed in the drop-down target selection list. This list contained the target names of all eCosPro and public eCos targets provided within eCosPro Developers Kit Releases and can be very long. To reduce this list a profile may restrict the targets displayed to only the targets with specific attributes. For example, for targets tested and supported within the 4.0.1 eCosPro Developers Kit Release, targets displayed may be restricted to those with the attribute `release_v4.0.1`. Similarly, the attribute `target_ecospro` may be used to limit targets displayed to those for which commercial support from eCosCentric is available.
	Currently only one attribute may be selected for filtering. Filtering may also be disabled by unchecking "`Filter targets according to attributes`" within the *Display* Tab of the *Settings* dialog View → Settings (**Ctrl+T**).

Creating or editing user-defined toolchains

The *Toolchain Edit* Dialog illustrated in Figure 11.6, "Toolchain editing dialog" provides a means by which user-defined toolchains can be added, edited or deleted from within eCosPro Profiles.

Figure 11.6. Toolchain editing dialog

Descriptions of the individual fields:

`Name`	The descriptive name used in the drop-down list of toolchains within the edit profile dialog.
`Description`	A textual description of the toolchain.
`Description`	A textual description of the toolchain.
`Prefix`	The GNU Toolchain prefix associated with this toolchain. The editor will look for executables of the form **<prefix>-gcc** to validate the toolchain path, and in the event a toolchain is deleted or uninstalled will use this prefix to match prospective alternative toolchains against a target when setting up the paths for a profile.
`Path`	An input field containing the name of the directory containing the GNU Toolchain executables **<prefix>-gcc**, **<prefix>-g++**, **<prefix>-ld**, **<prefix>-gdb** and so on.

The button to the right of the input field will allow you to navigate to and select the directory through the host's native interface rather than typing the directory name in the input field. |

The `Delete` button will delete an existing user-defined toolchain, `Cancel` will undo any changes made and `OK` will accept the changes.

eCos Configuration Tool Documents

Configuration Save File

eCos configuration settings and other information (such as disabled conflicts) that are set using the eCos Configuration Tool are saved to a file between sessions. By default, when the eCos Configuration Tool is first invoked, it reads and displays information from the Component Registry and displays the information in an untitled blank document. You can perform the following operations on a document:

Save the currently active document

Use the File → Save (**Ctrl+S**). menu item or click the *Save Document* icon on the toolbar; if the current document is unnamed, you will be prompted to supply a name for the configuration save file.

Figure 11.7. Save As dialog box
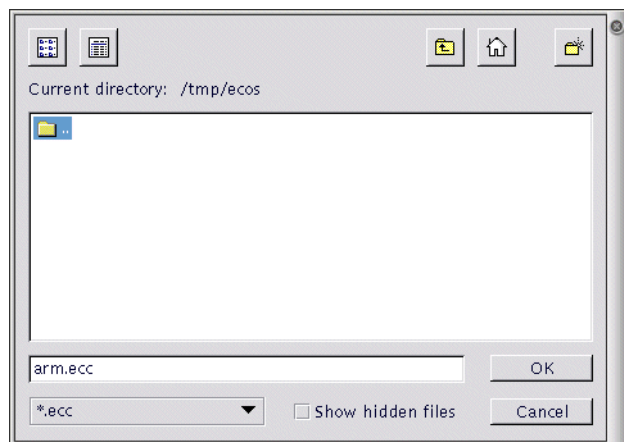
> **Note**
>
> If there are no conflicts within the eCos configuration and the *Check for conflicts: Before saving configuration* option within the *Conflict Resolution* tab is selected (see the section called "Resolving conflicts"), the configuration tool will automatically regenerate the build tree.

Open an existing document

Select File → Open (**Ctrl+O**) or click the *Open Document* icon on the toolbar. You will be prompted to supply a name for the configuration save file.

Figure 11.8. Open dialog box
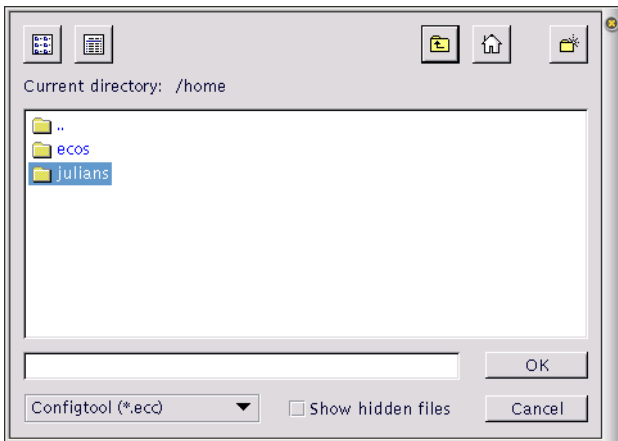
Open a document you have used recently

Click its name at the bottom of the *File* menu.

Documents may also be opened by:

• double-clicking a Configuration Save File in the desktop explorer (Windows only);

• invoking the eCos Configuration Tool with the name of a Configuration File as command-line argument, or by creating a shortcut to the eCos Configuration Tool with such an argument (under Windows or a suitable Linux desktop environment).

Create a new blank document based on the Component Registry or Profile and default target and template

Select File → New (`<target>:<template>`) (**Ctrl+N**) or click the *New Document* icon on the toolbar.

Create a new blank document based on the Component Registry or Profile *and* a different target or template

Select File → New Target (**Alt+N**).

Save to a different file name

Select File → Save As.... You will be prompted to supply a new name for the configuration save file.

⚠️ **Caution**

If there are no conflicts within the eCos configuration and the *Check for conflicts: Before saving configuration* option within the *Conflict Resolution* tab is selected (see the section called "Resolving conflicts", the configuration tool will automatically regenerate the build tree.

Build and Install Trees

The location of the build and install trees are derived from the eCos save file name as illustrated in the following example:

Save file name = "c:\My eCos\config1.ecc"

Install tree folder = "c:\My eCos\config1_install"

Build tree folder = "c:\My eCos\config1_build"

These names are automatically generated from the name of the save file.

See also Chapter 22, *CDL Concepts*.

Chapter 12. Getting Help

The eCos Configuration Tool contains several methods for accessing online help.

Context-sensitive Help for Dialogs

Most dialogs displayed by the eCos Configuration Tool are supplied with context- sensitive help. You can then get help relating to any control within the current dialog box by

• Right-clicking the control (or pressing **F1**)

  A "What,s This?" popup menu will be displayed. Click the menu to display a brief description of the function of the selected control.

• Clicking the question mark icon in the dialog caption bar (Windows) or the question mark button on the dialog (Linux).

  A question mark cursor will be displayed. Click on any control to display a brief description of its function.

Some dialogs may have a *Help* button. You can press this to display a more general description of the function of the dialog box as a whole. This help will be in HTML form; for more information, see below.

Context-sensitive Help for Other Windows

In the *Help* menu, click *Help On...* and then click on a window (or click on the arrow/question mark button on the toolbar, then click on a window). A small popup window page describing the window will be displayed. The same thing can be achieved by right-clicking on a window and clicking on *What's This?*.

Context-sensitive Help for Configuration Items

In the configuration window, right-click on a configuration item (or use **Shift**+**F10**). A context menu will be displayed; select *Visit Documentation* to display the page in the eCos documentation that most closely corresponds to the selected item.

Methods of Displaying HTML Help

1. The new default and recommended method to view help is through the Eclipse help system which is opened by the eCos Configuration Tool. This will display a contents hierarchy on the left and HTML pages on the right; see Figure 12.1, "Eclipse Help". Documentation for all eCosPro Developer's Kit components may be found herein, including documentation of all the eCosPro components installed locally. The documentation is fully searchable, with the search index generated automatically on the first search.

Figure 12.1. Eclipse Help
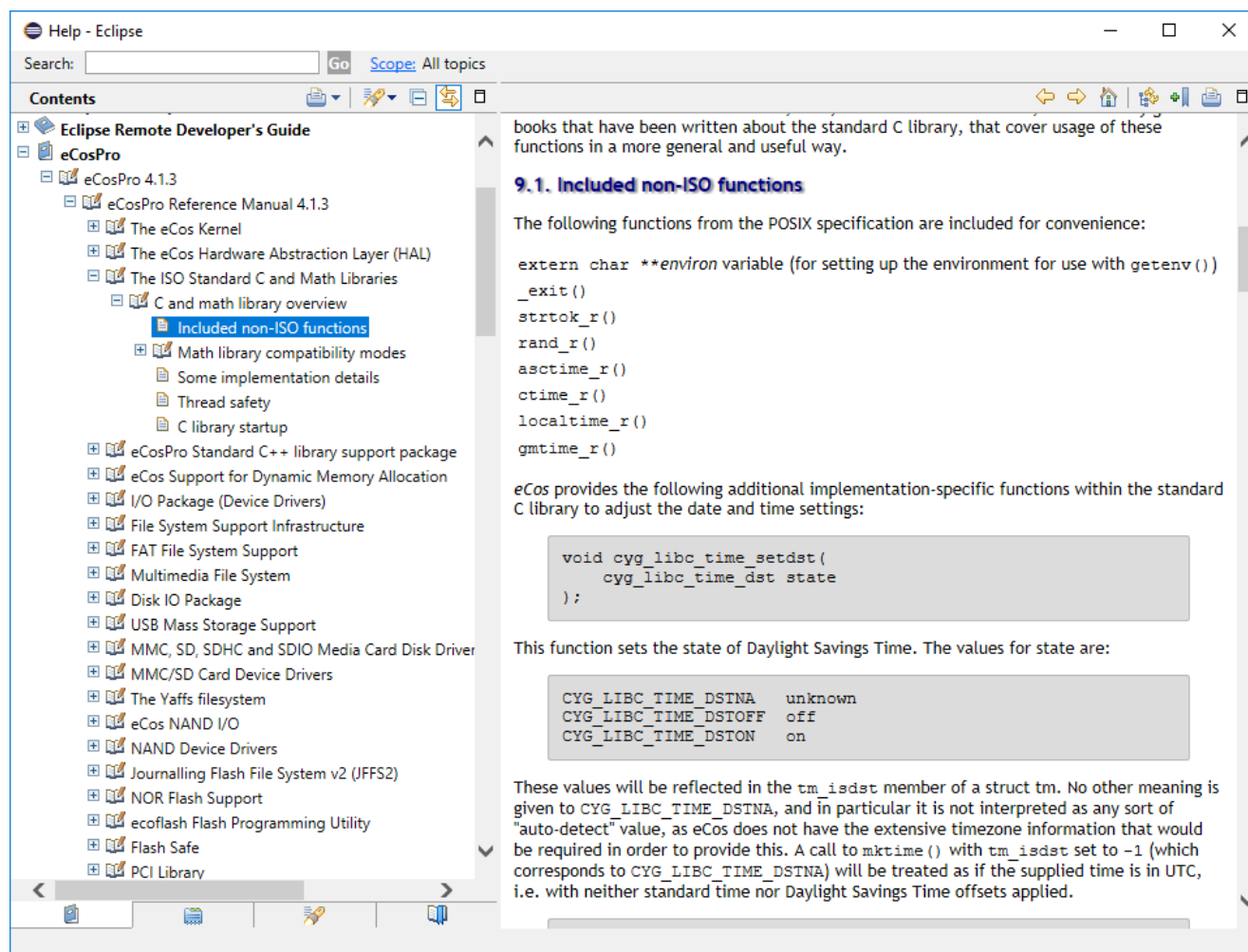
On each new installation of an eCosPro release, the Eclipse Help search cache is flushed to force Eclipse to re-generate it's help search index. This is also flushed using the menu item: Help → Rebuild eCos Documentation

If you have run an older version of the eCos Configuration Tool on your development host, the previous default or selected method of displaying help will be used. This can now be switched to Eclipse help as follows:

a. Within the eCos Configuration Tool open the settings dialog: View → Settings (**Ctrl+T**)

b. Select the `Viewers/Shell` tab

c. Select the radio button alongside `Eclipse help` under the `View documentation using:` heading.

d. Press the OK button to close the `Settings dialog`.

2. Using the default HTML browser. On Unix, if you do not already have an associated viewer for .html files, you can add one by using an entry in the .mailcap file in your home directory similar to this:

```
text/html; firefox %s
```

3. Using the specified browser.

4.

Warning

This method is deprecated

Using the deprecated internal help system. This will show an internal viewer similar to Microsoft HTML Help, with a contents hierarchy on the left and HTML pages on the right. The index is regenerated for each repository. If the documentation in the repository has changed but the contents does not reflect this, please use the Help → Rebuild eCos Documentation menu item.

If you wish, you may choose to have *HTML Help* displayed in a browser of your choice. To do this, select View → Settings (**Ctrl +T**) and use the controls in the View Documentation group to select the replacement browser. Note that the Navigation facilities of the built-in *HTML Help* system will be unavailable if you choose this method of displaying help.

Chapter 13. Customization

The following visual aspects of the eCos Configuration Tool can be changed to suit individual preferences. These aspects are saved on a per-user basis, so that when the eCos Configuration Tool is next invoked by the same user, the appearance will be as set in the previous session.

Window Placement

The relative sizes of all windows in the eCos Configuration Tool may be adjusted by dragging the splitter bars that separate the windows. The chosen sizes will be used the next time the eCos Configuration Tool is invoked by the current user.

All windows except the *Configuration Window* may be shown or hidden by using the commands under the *View* menu (for example, View → Output (**Alt+2**) or the corresponding keyboard accelerators (**Alt+1** to **Alt+4**).

Your chosen set of windows (and their relative sizes) will be preserved between invocations of the eCos Configuration Tool.

Settings

To change other visual aspects, select View → Settings (**Ctrl+T**) and then select the *Display* and *Viewers* tabs depending on the settings you wish to alter.. The options are as follows:

Settings: Display tab

Figure 13.1. Settings dialog, Display tab
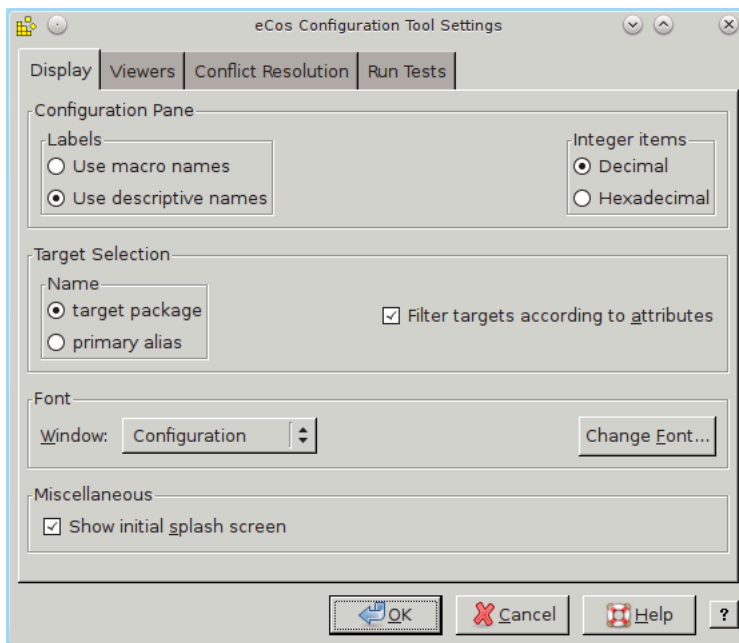
Labels

In the configuration window, you can choose to have either *descriptive names* (the default) or *macro names* displayed as tree item labels. Descriptive names are generally more comprehensible, but macro names are used in some contexts such as conflict

---

resolution and may be directly related to the source code of the configuration. Note that it is possible to search for an item in the configuration view by selecting Edit → Find (**Ctrl+F**) (see Chapter 16, *Searching*). Descriptive, item and macro names can be searched, as well as descriptions and current and default values.

Integer Items

You can choose to have integer items in the Configuration Window displayed in decimal or hexadecimal format.

Target Selection

When selecting a target for a configuration or for test execution, the name of the target in the drop down lists may either be the name of the *"target package"* or the *"primary alias"* of the target package (a descrptive name).

Early versions of the configuration tool used the descriptive name in this drop down list, but the reference guide mostly refers to targets by their package name. This was found to be confusing for some developers so an option was introduced to allow the target to be selected by either the original method (the *"primary alias"*) or by the *"target package"* name.

Since the list of targets can be very long, making the selection of a target both difficult and confusing, the option to *"Filter targets according to attributes"* allows the developer to restrict the drop-down list to the target attributes set within the active profile. Checking this box will restrict the targets displayed according to the attributes of the target package and the *"Target Filter"* set by the active profile. For more information see the section called "Creating or editing user profiles".

Font

Change the font for a particular window by selecting the window name using the drop-down list, then clicking on *Change Font* to select a font for that window. The changes will be applied when the press *OK* to dismiss the Settings dialog. If you never make font changes, then the windows will take the default setting determined by your current Windows or Unix environment.

Miscellaneous

If the *Splash Screen* checkbox is checked, a *splash* window will appear as the application is loading for the first three occassions the application is run, and for the first three occassions each time the application is upgraded. Uncheck this to eliminate the splash screen from ever being displayed.

Settings: Viewers tab

Figure 13.2. Settings dialog, Viewers tab
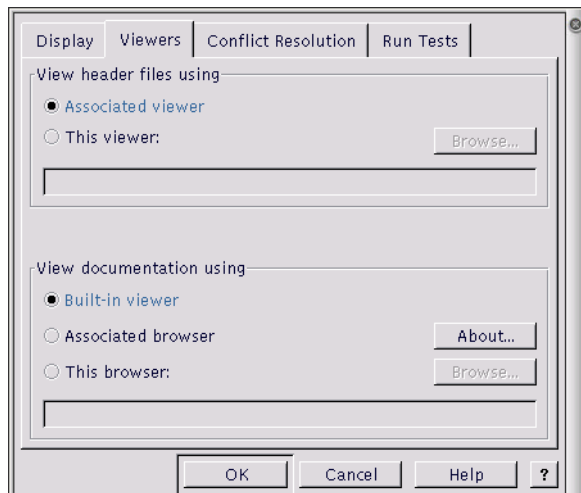
View header files

You can change the viewer used to display header files.

View documentation

You can change the viewer used to display HTML files. See the section called "Methods of Displaying HTML Help".

Chapter 14. Screen Layout

The following windows are available within the eCos Configuration Tool:

- Configuration Window

- Properties Window

- Short Description

- Conflicts

- Output

The layout of the windows may be adjusted to suit your preferences: see the section called "Settings".

Configuration Window

This is the principal window used to configure eCos. It takes the form of a tree-based representation of the configuration items within the currently loaded eCos packages.

In the case of items whose values may be changed, controls are available to set the item values. These either take the form of check boxes or radio buttons within the tree itself or cells to the right of the thin vertical splitter bar. Controls in the tree may be used in the usual way; cells, however, must first be activated.

To activate a cell, simply click on it: it will assume a sunken appearance and data can then be edited in the cell. To terminate in-cell editing, click elsewhere in the configuration window or press **ENTER**. To discard the partial results of in-cell editing and revert to the previous value, press **ESC**.

Figure 14.1. Configuration Window
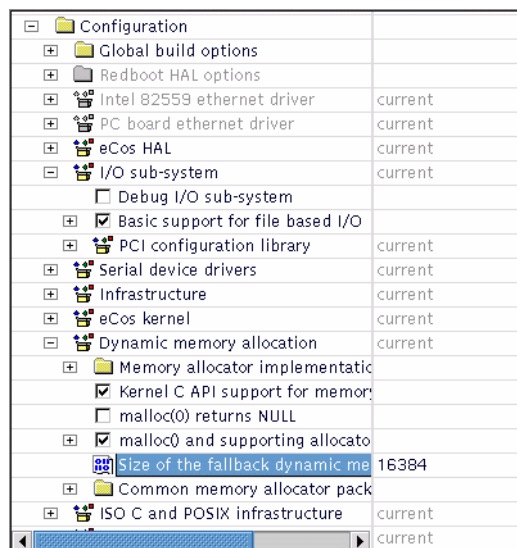
Cells come in three varieties, according to the type of data they accept:

Table 14.1. Cell types

Cell Type	Data Accepted
Integer	Decimal or hexadecimal values

Cell Type	Data Accepted
Floating Point	Floating point values
String	Any

In the case of string cells, you can double-click the cell to display a dialog box containing a larger region in which to edit the string value. This is useful in the case of long strings, or those spanning multiple lines.

Disabled items

Some items will appear disabled. In this case the item label and any associated controls and cells will be grayed. It is not be possible to change the values ofdisabled items.

Right-Clicking

You can right-click on an item in the configuration window item to display a pop-up menu which (depending on the type of the item selected) allows you to:

- *Properties* – information relating to the currently selected item is displayed. The information is equivalent to that displayed in the Properties Window.

- *Restore Defaults* - the default value of the currently selected item is restored.

- *Visit Documentation* - causes the HTML page most closely relating to the currently selected item to be displayed. This has the same effect as double-clicking the URL property in the Properties Window.

- *View Header File* – this causes the file containing the items to be displayed. This is equivalent to double-clicking on the File property in the Properties Window. The viewer used for this purpose may be changed using the View → Settings (**Ctrl+T**) menu item (see the section called "Settings"). Note that this operation is only possible when the current configuration is saved, in order to avoid the possibility of changing the source repository.

- *Unload Package* - this is equivalent to using the Build → Packages (**Ctrl+P**) menu item to select and unload the package in question.

Conflicts Window

This window exists to display any configuration item conflicts. Conflicts are the result of failures to meet the requirements between configuration items expressed in the CDL. See the section called "Conflicts"

Figure 14.2. Conflicts Window

The window comprises three columns:

- *Item*

  This is the macro name of the first item involved in the conflict.

- *Conflict*

This is a description of the conflict type. The currently supported types are "unresolved", "illegal value", "evaluation exception", "goal unsatisfied" and "bad data".

• *Property*

This contains a description of the configuration item,s property that caused the conflict.

Within the conflicts window you can right-click on any item to display a context menu which allows you to choose from one of the following options:

To locate the item involved in the conflict, double-click in the first or third column, or right-click over the item and choose *Locate* from the popup menu.

You can use the Tools → Resolve Conflicts. menu item, or right-click over the item and select *Resolve* from the popup menu, to resolve conflicts — the section called "Resolving conflicts".

Output Window

This window displays any output generated by execution of external tools and any error messages that are not suitable for display in other forms (for example, as message boxes).

Within the output window you can right-click to display a context menu which allows you to:

• Save the contents of the window to a file

• Clear the contents of the window

Properties Window

This window displays the CDL properties of the item currently selected in the configuration window. The same information may be displayed by right-clicking the item and selecting "properties".

If a property is displayed with a light-pink background, this inicates that the property is in conflict with another setting in the current configuration.

Figure 14.3. Properties Window

Property	Value
URL	redirect/the-ecos-hardware-abstraction-layer-hal.html
File	/tmp/untitled_install/include/pkgconf/hal_i386.h
Macro	CYGPKG_HAL_I386
Value	current
Default	current
Parent	CYGPKG_HAL
Hardware	
IncludeDir	cyg/hal
DefineHeader	hal_i386.h
Compile	hal_misc.c context.S i386_stub.c hal_syscall.c
Make	<PREFIX>/lib/target.ld: <PACKAGE>/src/i386.ld $(CC) -E -P -\

The following properties may be double-clicked as follows:

• *URL* – double-clicking on a URL property causes the referenced HTML page to be displayed. This has the same effect as right-clicking on the item and choosing "Visit Documentation".

- *File* – double-clicking on a File property in a saved configuration causes the File to be displayed. The viewer used for this purpose may be changed using the View → Settings (**Ctrl+T**) menu item. Note that this operation is only possible when the current configuration is saved, in order to avoid the possibility of changing the source repository.

- *Requires* – double-clicking on this property will Locate the first property displayed in the value field.

If a property value is selected with a Secondary mouse click, you are given the option of copying the property value to the desktop's cut-and-paste buffer. This allows you to either use the value when developing your own package or to paste it into the search dialog to determine additional information regarding that package.

Short Description Window

This window displays a short description of the item currently selected in the configuration window. More extensive documentation may be available by right-clicking on the item and choosing "Visit Documentation".

Chapter 15. The Configuration

Adding and Removing Packages

To add or remove packages from the configuration, select Build → Packages (**Ctrl+P**). The following dialog box will be displayed:

Figure 15.1. Packages dialog box
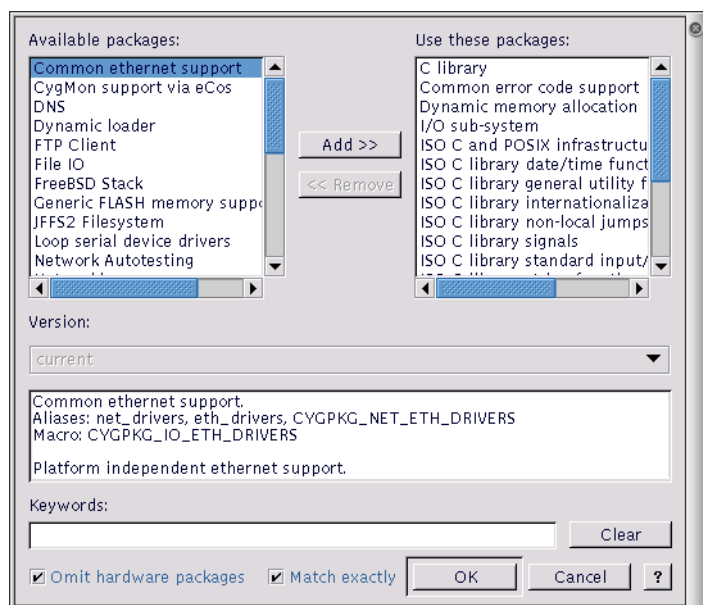
The left-hand list shows those packages that are available to be loaded. The right-hand list shows those that are currently loaded. In order to transfer packages from one list to another (that is, to load or unload packages) double-click the selection or click the *Add* or *Remove* buttons.

The version drop-down list displays the versions of the selected packages. When loading packages, this control may be used to load versions other than the most recent (current). Note that if more than one package is selected, the version drop-down list will display only the versions common to all the selected packages.

The window under the version displays a brief description of the selected package. If more than one package is selected, this window will be blank.

Under the description window there is a *Keywords* control into which you can type a string to be matched against package names, macro names and descriptions. The lists are updated a second or so after typing has stopped. If you type several separate words, all of these words must be associated with a given package for that package to be displayed. If you select the *Match exactly* checkbox, then the string is taken to be a complete fragment and matched against the beginning of a name, macro name or descriptions. All matches are done case-insensitively.

If you check *Omit hardware packages*, only non-hardware packages will be shown.

Switching Targets, Repositories and Versions

To migrate an eCos configuration to a different target, version of eCosPro, or repository, the following steps should be followed:

• Export the existing configuration to an `.ecm` file.

- Confirm the target and template set in the existing configuration using the *Repository Information* dialog illustrated in Figure 15.7, "Repository and Configuration Information dialog box". Help → Repository Information (**Ctrl+I**)

- If applicable, switch to the new version, repository or profile using the file menu: Build → Profile (**Alt+P**).

- Create a new configuration by selecting the new target (if applicable) and template, selecting same template as the original configuration File → New Target (**Alt+N**) illustrated in Figure 7.3, "Target and Template selection".

  Set the hardware template and the packages template to the originals determined above.To select a hardware template, choose from the first drop-list. To choose a packages template, choose from the second. Brief descriptions of each kind of template are provided in the corresponding edit boxes.

- Import the original configuration from the exported `.ecm` file create in the first step above.

- Resolve conflicts, if any.

- Save the configuation.

> **Note**
>
> - While it remains possible to switch targets within a configuration using the command line tool **ecosconfig**, users are advised against this as inferred changes within common hardware packages to both the targets are not reset or re-inferred unless a conflict arises as a result of the switch. This may result in unexpected or unintended consequences. For example, while a template may contain the SPI package, this package may be disabled through inference as a result of the initial target hardware lacking an SPI device. However, when switching to a processor variant with more cells, such as one that includes an SPI device, the SPI package could remain disabled when the configuration is switched to the more superior variant (unless of course it is marked as required by the CDL which would result in a conflict and so would be re-enabled). While this is perfectly legimate behavior (e.g. SPI support could be disabled on a superior variant if it were not required by the CDL), users are advised to use the method above when switching targets.
>
> - The same method described above may be used to switch an existing target, template and custom configuration to a new version, profile or repository.

Platform Selection

To add, modify or remove entries in the list of platforms used for running tests, select Tools → Platforms (**Ctrl+P**). The following dialog will be displayed:

Figure 15.2. Platforms dialog box
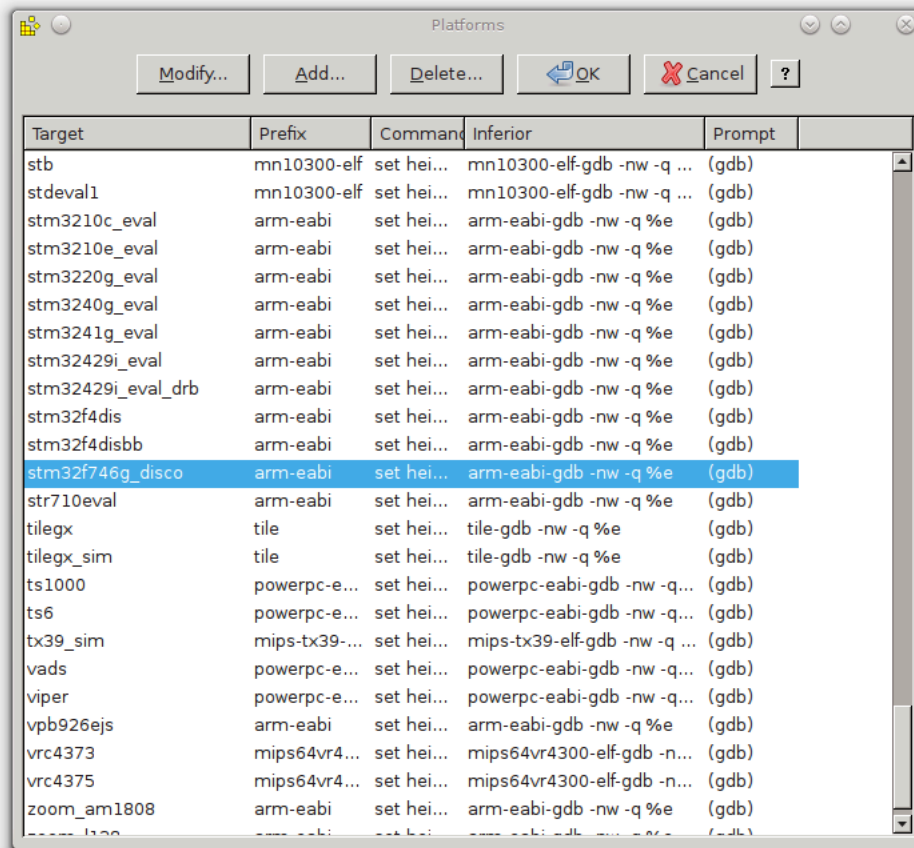
You may add, modify or remove platform entries as you wish, but in order to run tests, a platform must be defined to correspond to the currently loaded hardware template. The information associated with each platform name is used to run tests.

To modify a platform, click the *Modify* button with the appropriate platform selected, or double-click on an entry in the list. A dialog will be displayed that allows you to change the command prefix, platform type and arguments for *GDB*.

Figure 15.3. Platform Modify dialog box

To add a new platform, click the *Add* button. A similar dialog will be displayed that allows you to define a new platform. To remove a platform, click the *Delete* button or press the **DEL** key with the appropriate platform selected.

Platform dialog fields

Command prefix   Used when running tests in order to determine the names of the executables (such as gdb) to be used. For example, if the gdb executable name is "arm-eabi-gdb.exe" the prefix should be set to "arm-eabi".

Arguments for GDB   Specifies the set of commands, in sequence, are normally required to run an eCos test from start to finish. This includes how to make provision for retrieving diagnostic messages from the target.

                          Lines that provide Platform Variables (see below) that have not been set within platform properties (see Figure 8.2, "Properties dialog box") will be omitted from the arguments passed to GDB.

Inferior   Specifies the full GDB command with arguments to be launched in order to commence each test. If the executable is not on the PATH, the full path to the executable must be used.

Prompt   Specifies the prompt to expect before executing the next gdb command provided in the list above.

Variables: To facilitate easy modification of test executables, timeouts, baud rates, target location, etc. the *Arguments for GDB* and *Inferior* may reference variables which are replaced at execution time. The following variables may be used:

Platform variables

%b   Serial baud rate (*Serial Baud* value set in Figure 8.2, "Properties dialog box")

%e	Full path name to the test executable
%p	Port, such as the serial port, path to the device, TCPIP address and port, pipe and pipe command (*Serial Port*, *TCP/IP Address* or *Pipe Command* values set in Figure 8.2, "Properties dialog box")
%r	Remote timeout (*Remote* value set under *Timeouts* in Figure 8.2, "Properties dialog box")
%E(*VARIABLE*)	The value of the environment variable *VARIABLE*.
	See the section called "Environment Variables" for further information on environment variables.

Using Templates

To switch a configuration to a new template, select Build → Templates (**Ctrl+M**).

The following dialog box will be displayed:

Figure 15.4. Templates dialog box
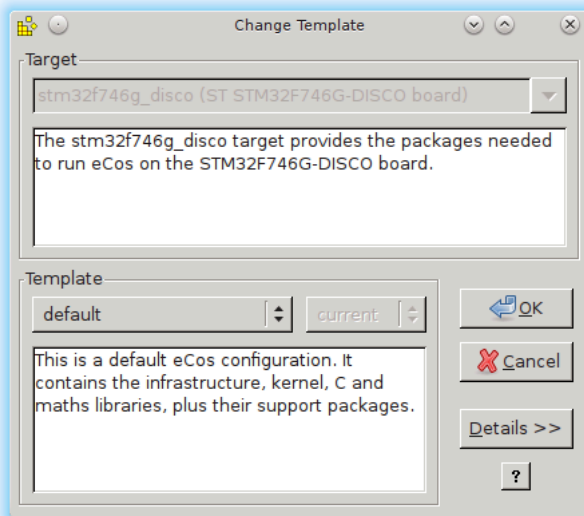
The target hardware dialog will be disabled as switching hardware within a configuration can have unintended consequences within eCos. See the section called "Switching Targets, Repositories and Versions" for details on how to switch a configuration to a new target.

To choose a packages template, choose from the template drop-down. Brief descriptions of each kind of template are provided in the corresponding edit boxes.

Resolving conflicts

During the process of configuring eCos it is possible that conflicts will be created. For more details of the meaning of conflicts, see Chapter 22, *CDL Concepts*.

The Conflicts Window displays all conflicts in the current configuration. Additionally, a window in the status bar displays a count of the conflicts. Because the resolution of conflicts can be time-consuming, a mechanism exists whereby conflicts can be resolved automatically.

You can choose to have a conflicts resolution dialog box displayed by means of the View → Settings (**Ctrl**+**T**) menu item, on the *Conflict Resolution* tab of the dialog.

Figure 15.5. Options
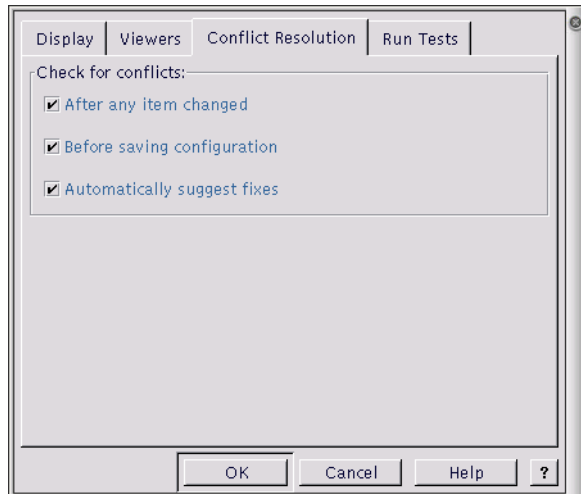
You can choose to have conflicts checked under the following circumstances:

- After any item is changed (in other words, as soon as the conflict is created)

- Before saving the configuration (including building)

- Never

The method you chose depends on how much you need your configuration to be free of conflicts. You may want to avoid having to clean up all the conflicts at once, or you may want to keep the configuration consistent at all times. If you have major changes to implement, which may resolve the conflicts, then you might want to wait until after you have completed these changes before you check for conflicts.

> **Note**
>
> If you choose to check conflicts after any item is changed, only newly arising conflicts are displayed. If you choose to check for conflicts before saving the configuration, the complete set is displayed.

Automatic resolution

If you check the "Automatically suggest fixes" check box, a conflicts resolution dialog box will be displayed whenever new conflicts are created. The same dialog box may be displayed at any stage by means of the Tools → Resolve Conflicts menu item.

The conflicts resolution dialog box contains two major windows.

Figure 15.6. Resolve conflicts window
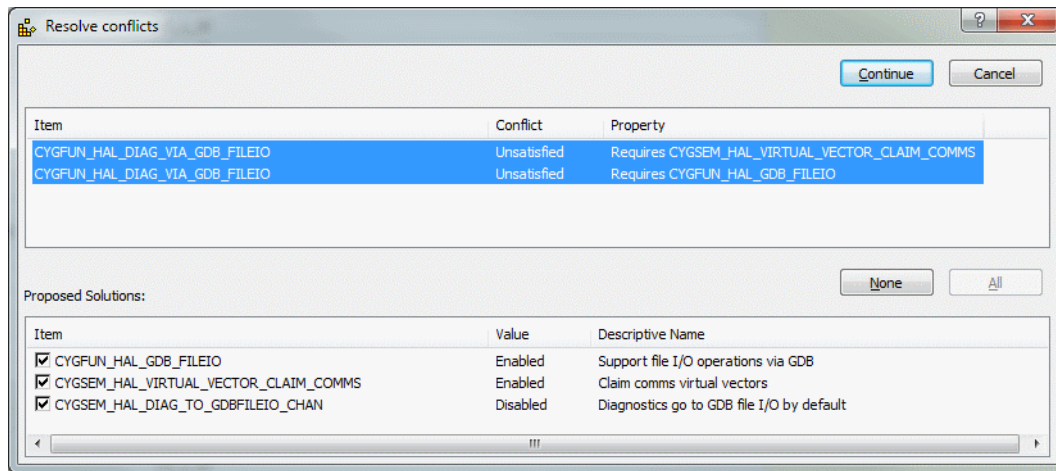
The upper contains the set of conflicts to be addressed; the format of the data being as that of the Conflicts Window. The lower window contains a set of proposed resolutions – each entry is a suggested configuration item value change that as a whole may be expected to lead to the currently selected conflict being resolved.

Note that there is no guarantee:

- that automatic resolutions will be determinable for every conflict.

- that the resolutions for separate conflicts will be independent. In other words, the resolution of one conflict may serve to prevent the resolution of another.

- that the resolution conflicts will not create further conflicts.

The above warnings are, however, conservative. In practice (so long as the number and extent of conflicts are limited) automatic conflict resolution may be used to good effect to correct problems without undue amounts of programmer intervention.

In order to select the conflicts to be applied, select or clear the check boxes against the resolutions for each proposed resolution. By default all resolutions are selected; you can return to the default state (in other words, cause all check boxes for each conflict to again become checked) by pressing the "Reset" button. Note that multiple selection may be used in the resolutions control to allow ranges of check boxes to be toggled in one gesture.

When you are happy to apply the selected resolutions for each conflict displayed, click *Apply*; this will apply the resolutions. Alternatively you may cancel from the dialog box without any resolutions being applied.

Configuration Information

An important part of creating products using eCos is determining the licenses under which the sources forming the active packages of an eCos configuration are distibuted. To assist with this process, all top-level eCos packages have one or more *license* attributes associated with the package. Each attribute name indicates the licenses under which the source code of the package is distributed. Some example attributes and their associated licenses are:

Table 15.1. License Attributes

Attribute	License
license_ecospro	eCosPro License

Attribute	License
license_ecospro_eval	eCosPro Evaluation License
license_ecosgpl	eCos Public License
license_gplv2	GNU GENERAL PUBLIC LICENSE v2
license_gplv3	GNU GENERAL PUBLIC LICENSE v3
license_bsd3clause	Modified BSD "2 clause" or "3 clause" Licenses
license_bsd4clause	Modified BSD "4 clause" Licenses
license_zlib	ZLIB License
license_eclipse	Eclipse Public License - v1.0
license_opl	Open Publication License
license_yaffsgpl	YAFFS GPL Dual license

If a top-level package is select within the configuration tree, the properties window will include an *Attributes* property field that contains the attributes associated with the package.

Figure 15.7. Repository and Configuration Information dialog box
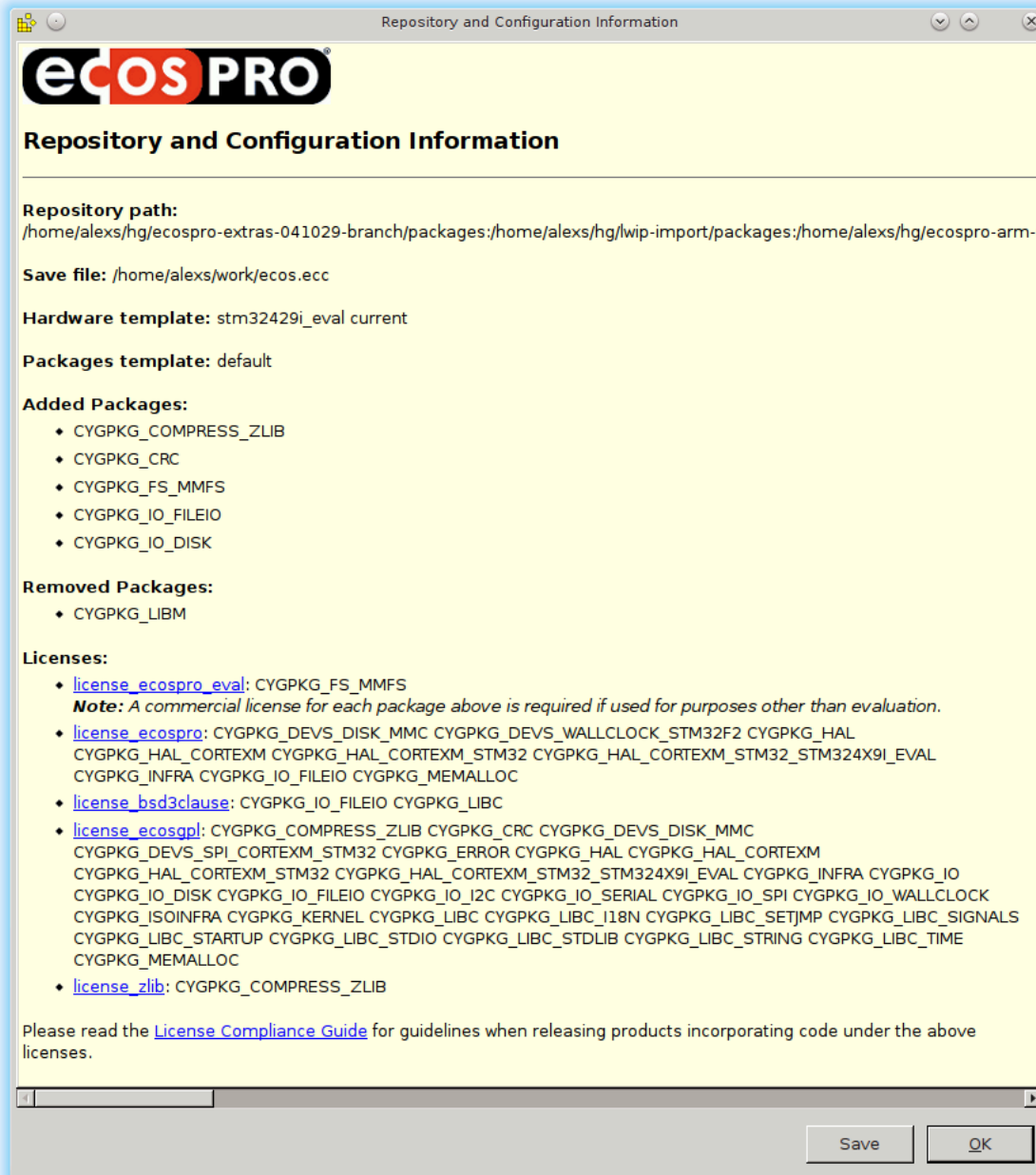
In addition to the *Attributes* property field, the configuration tool includes a *Repository and Configuration dialog*, as illustrated above. This dialog displays summary information of the user's configuration and may be opened by selecting the Help→Repository Information (**Ctrl+I**) menu option. An HTML copy of the summary information may also be saved to a local file for reference purposes through the *Save* option of the dialog box.

As illustrated, the repository, hardware template and packages template used in generating the initial configuration are displayed, as well as any packages subsequently added or removed from the initial configuration. Lastly, the license attributes allocated to each top-level package used in the configuration are presented, with the macro names of packages with that attribute appearing beneath each license attribute. If a license attribute is selected, the configuration tool will open a browser window to the corresponding

license on eCosCentric's web site. If an evaluation package has been included, the developer is alerted in the summary that binaries generated by this eCos configuration will require a commercial license for the eCosPro packages listed. Note that because packages may have multiple license attributes and so the macro names may appear more than once in the licensing breakdown.

Note

The *Attributes* property as well as the additional repository, configuration and licensing information are only available in verions of eCosPro 4.0 and above.

Chapter 16. Searching

Select Edit → Find (**Ctrl+F**). You will be presented with a Find dialog box:

Figure 16.1. Find dialog box
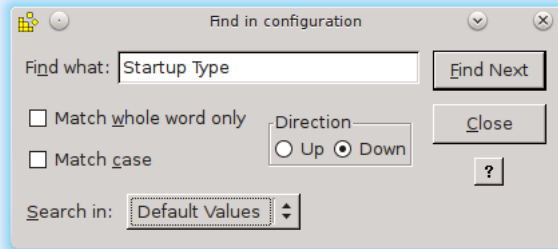
Using this dialog box you can search for an exact text string in any one of five ways, as specified by your selection in the "Search in" drop-list:

- Macro names - the search is for a text match within configuration item macro names

- Item names - the search is for a text match within configuration item descriptive names

- Short descriptions - the search is for a text match within configuration item short descriptions

- Current values - the search is for a text match within the current value of the configuration items

- Default values - the search is for a text match within the default value of the configuration items

Note that to invoke *Find* you can also click the *Find* icon on the toolbar.

Chapter 17. Building

When you have configured eCos, you may build the configuration.

On the *Build* menu, click:

- *Library* (or click the Build Library icon on the toolbar) – this causes the eCos configuration to be built. The result of a successful build will be (among other things) a library against which user code can be linked

- *Tests* – this causes the eCos configuration to be built, and additionally builds the relevant test cases linked against the eCos library

- *Clean* – this removes all intermediate files, thus causing a subsequent build/library or build/tests operation to cause recompilation of all relevant files.

- *Stop* – this causes a currently executing build (any of the above steps) to be interrupted

- *Generate Build Tree* – this causes a build tree to be generated for the existing eCos configuration. The option is greyed out if there are any conflicts. This will only overwrite makefiles and headers that must be changed as a result of a change in the eCos configuration since the previous build tree was last generated.

Note

Any changes made to automatically generated makefiles and headers will be lost whenever a new build tree is generated. These files must therefore never be edited.

When either Build → Library (**F7**) or Build → Tests (**Shift**+**F7**) are selected, before the build is permitted to start:

- if there are any conflicts within your eCos configuration, you will be prompted to resolve these

- if you have not already saved your eCos configuration, you will be prompted to do so

- a build tree will be generated automatically. This will only overwrite makefiles and headers that must be changed as a result of a change in the eCos configuration since the previous build tree was last generated.

Build options may be displayed by using the Build → Options menuitem. This displays a dialog box containing a drop-list control and two windows. The drop-list control allows you to select the type of build option to be displayed (for example "LDFLAGS" are the options applied at link-time. The left-hand window is a tree view of the packages loaded in the current configuration. The right-hand window is a list of the build options that will be used for the currently selected package.

Note that this dialog box currently affords only read-only access to the build options. In order to change build options you must edit the relevant string configuration item.

A single level of inheritance is supported: each package,s build options are combined with the global options (these are to be found in the "Global build options" folder in the configuration view).

Figure 17.1. Build Options
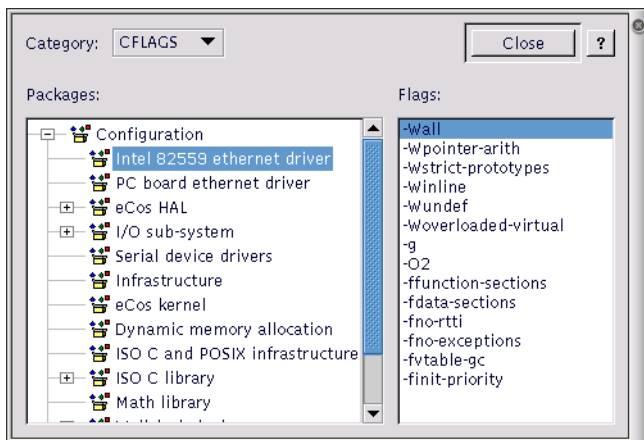
Selecting Build Tools

Normally the profile or installation process will supply the information required for the eCos Configuration Tool to locate the build tools (compiler, linker, etc.) necessary to perform a build. However if this information is not registered, or it is necessary to override or specify the location manually (for example, to use a user-defined toolchain), select Tools → Paths → Build Tools... to set user-defined build tools. This location will take precedence over the build tools defined by the *Profile* or PATH environment. This setting will also remain persistent on subsequent invocations of the eCos Configuration Tool. The following dialog box will be displayed:

Figure 17.2. Build tools
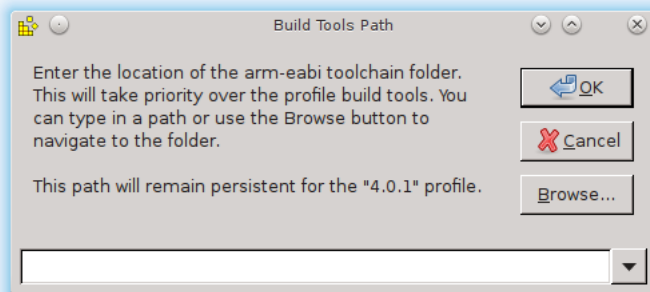
This dialog box allows you to locate the folder containing the build tools or toolchain.

> **Note**
>
> User-defined build tools set in this way will override the the build tool setting provided by the *Profile*. Furthermore this override will be saved persistently for that profile. However if you switch to a different profile, then the appropriate build tool settings for that new profile will be used instead (or if user-defined build tools had previously been set for that new profile, then those will be used).

Selecting Host Tools

Normally the installation process will supply the information required for the eCos Configuration Tool to locate the host tools (cat, ls, etc.) necessary to perform a build. However if this information is not registered, or it is necessary to specify the location

manually (for example, when a new toolchain installation has been made), select Tools → Paths → Host Tools.... The following dialog box will be displayed:

Figure 17.3. Host tools
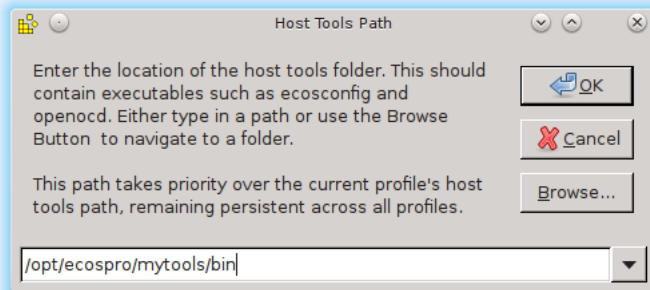
Note

If the host tools are set here, they will take precedence over the host tools specified by the profile or PATH environment. This setting will also remain persistent across all profiles and on subsequent invocations of the Configuration Tool.

Chapter 18. Execution

Test executables that have been linked using the Build/Tests operation against the current configuration can be executed by selecting Tools → Run Tests (**Ctrl+F5**) which will result in the dialog illustrated in Figure 18.1, "Run tests" being shown.

When this dialog is active, the Configuration Tool looks for a platform name corresponding to the currently loaded hardware template. If no such platform is found, a dialog will be displayed for you to define one; this dialog is similar to that displayed by the *Add* function in the Tools → Platforms... dialog, but in this case the platform name cannot be changed.

Figure 18.1. Run tests
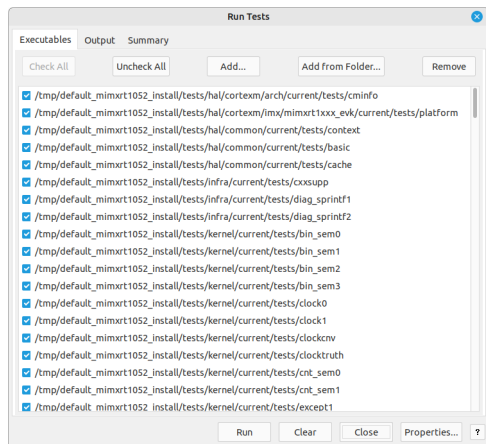
Note

This dialog resizable.

This dialog comprises of three tabs: *Executables*, *Output* and *Summary*.

Four buttons appear on the dialog itself: *Run/Stop*, *Clear*, *Close* and *Properties*.

The *Run* button is used to initiate a test run. Those tests selected on the *Executables* tab are run, and the output recorded on the *Output* and *Summary* tabs. During the course of a run, the *Run* button changes to *Stop* and the *Clear* and *Properties* buttons are disabled. The *Stop* button may be used to interrupt a test run at any point. While tests are running, and after any test run, the status bar to the left of the *Run/Stop* button will contain the total tests executed as well as the totals of each test result.

The *Clear* button is used to clear the output recorded on the *Output* and *Summary* tabs, as well as the status bar.

The *Close* button is used to close the dialog.

The *Properties* button is used to open the *eCos Configuration Tool Settings* dialog as described in the next section.

Properties

The *Properties* button is used to change the connectivity properties for the test run and when pressed will bring up the dialog illustrated in Figure 18.2, "Run Tests Properties dialog box".

Figure 18.2. Run Tests Properties dialog box
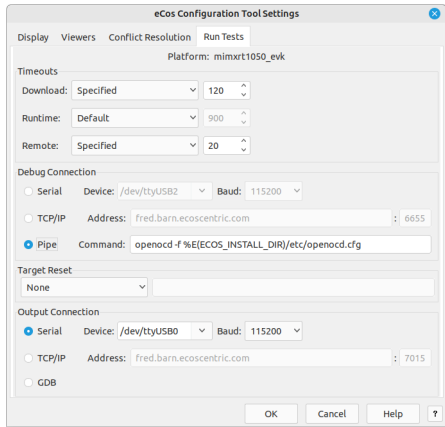
Download Timeout

This group of controls serves to set the maximum time that is allowed for downloading a test to the target board. If the time is exceeded, the test will be deemed to have failed for reason of "Download Timeout" and the execution of that particular test will be abandoned. This option only applies to tests run on hardware, not to those executed in a simulator. Times are in units of elapsed seconds.

Three options are available using the drop-down list:

- Calculated from file size - an estimate of the maximum time required for download is made using the (stripped) executable size and the currently used baud rate

- Specified - a user-specified value may be entered in the adjacent edit box

- None - no maximum download time is to be applied.

Run time Timeout

This group of controls serves to set the maximum time that is allowed for executing a test on the target board or in a simulator. If the time is exceeded, the test will be deemed to have failed for reason of "Timeout" and the execution of that particular test will be abandoned. In the case of hardware, the time is measured in elapsed seconds: in the case of a simulator it is in CPU seconds.

Three options are available using the drop-down list:

- None - no maximum download time is to be applied.

- Specified - a user-specified value may be entered in the adjacent edit box

- Default - a default value of 30 seconds is used

Debug Connection

The *Debug Connection* control is used to specify how the target board is to be accessed.

If the target board is connected using a serial cable, the *Serial* radio button should be checked. In this case you can select a port (On Windows COM1, COM2, … or on Linux specify the path to the serial device `/dev/ttyUSB0`, `/dev/ttyS0`) and an appropriate baud rate using drop-list boxes.

If the target board is accessed remotely using GDB remote protocol, the "TCP/IP" radio button should be checked. In this case you can select a valid host name or IP address and TCP/IP port number using edit boxes.

If the target board is accessed remotely using a command PIPE, the "Pipe" radio button should be checked and the command, including arguments, entered into the corresponding *Command* field. Do not include the Pipe "|" symbol in the *Command* - this is added by the eCos Configuration Tool.

Target Reset

The *Target Reset* control is used to specify how the target board is to be made ready (reset) before each test execution.

Three options are available using the drop-down list:

None   The test may be executed on the target hardware immediately, no hardware reset is required. For example, on a simulator or when the board is connected to a hardware debugger that resets or programs the target according to its configuration when a **GDB** client connects to it.

When using a *Pipe* command to **openocd**, openocd normally will reset and bring up the hardware in accordance with its scripts before accepting **GDB** commands from the *Pipe*. Similarly establishing a remote TCP/IP connection to a JTAG debugger such as the Ronetix PEEDI will normally configure and reset the hardware accordingly.

Command   Target hardware can be reset through the local host execution of a command or script. You will be required to provide the command, complete with arguments, in the field alongside. Do not include the pipe ("|") symbol. If the command is not on the path, the full path to the command must be entered. The command must also terminate with a zero exit code once the target hardware has been successfully reset. A non-zero exit code indicates the target as having failed and all following test executions are cancelled.

All output from the command will be displayed in the *Output* tab of the *Run dialog*.

Manual   The target requires manual or physical intervention (such as a power cycle or pressing the reset button). The user is directed through a dialog Window to reset the target hardware and press *OK* when the target hardware is ready to execute a test or *Cancel* to terminate the execution of the selected tests.

Output Connection

The *Output Connection* control is used to specify where any output of tests may be accessed:

GDB   This is the default and is typically used when tests are loaded and executed from RAM via a debug monitor such as RedBoot. This option may also be used with hardware debuggers when eCos isconfigured with the `CYG-FUN_HAL_DIAG_VIA_GDB_FILEIO` and `CYGFUN_HAL_GDB_FILEIO` configuration options enabled.

However, this method of extracting output is invasive as the target will cease test execution temporarily, either through the monitor or the hardware debugger, while the output is extracted through **GDB** to the host. In addition, when the target is configured with SMP, RedBoot-based debugging is currently not supported.

Serial   Select this radio button and choose the serial port on the host, along with the baud rate, on which diagnostic output from the target running eCos can be received. All data received on this port will appear in the *Output* tab described in the section called "Output Tab" and be interpreted as test output.

Many ARM target platforms include a Communication Device Class (CDC) USB Device using the sub-class Abstract Control Model (ACM). This enables input/output between the target and an external host to be configured in eCos through the USB-Serial connection. In addition, other targets include connectors for TTL to USB Serial converters to be attached, also providing a channel for input/output functionality between the target and an external host.

eCos supports configuring diagnostic output, such as test results, to be sent over a USB-Serial connection to a host. For example, the eCos configuration option `CYGHWR_HAL_CORTEXM_DIAGNOSTICS_INTERFACE` is available on certain targets to allow you to specify where diagnostic output is to be sent.

TCP/IP   Select this radio button and choose the host name or IPv4 address, as well as the port number, on which diagnostic output from the target running eCos can be received. All data received on this port will appear in the *Output* tab described in the section called "Output Tab" and be interpreted as test output.

This feature is especially useful for accessing remotely located target hardware over a network where the target is providing diagnostic output over a serial port attached to either a terminal server such as the IOLAN+ Terminal Servers, or the target is connected to a remote host which provices access to its serial port through Linux applications such as ser2net or Windows applications such as realterm or TCP-Com.

Executables Tab

This is used to adjust the set of tests available for execution. A check box against each executable name indicates whether that executable will be included when the *Run* button is pressed. The *Check All* and *Uncheck All* buttons may be used to check or uncheck all items. Items may also be individually checked or unchecked.

When the dialog is first displayed, it will be pre-populated with those test executables that have been linked using the Build/Tests operation against the current configuration.

Figure 18.3. Run tests
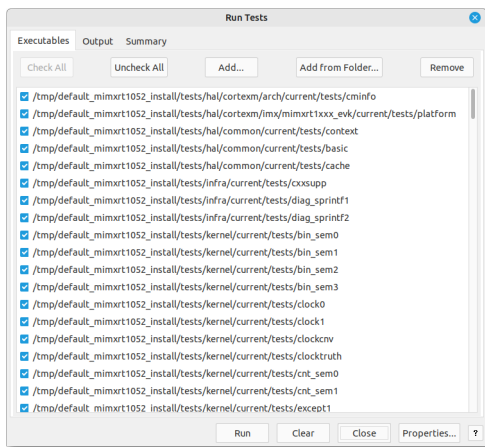
Highlighting items in the list and clicking the *Remove* button will remove those executables highlighted. Clicking the *Add* button will display a dialog box that allows you to add to the set of items. Equivalently the *Add from folder..* button may be used to add add a number of executables in a specified folder (optionally including subfolders, if you click on *Yes* when asked).

Figure 18.4. Add files from folder
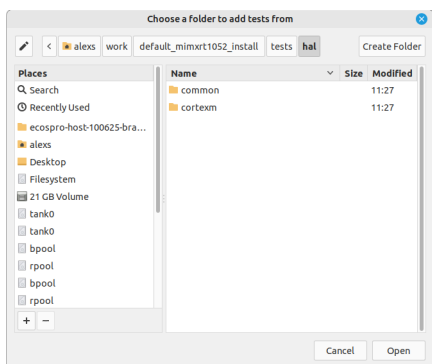
Output Tab

This tab is used to display the output from running tests. The output can be selected and copied by means of the popup menu displayed when you right-click in the window. The output is enriched to style test names as headings to aid with identification of individual test output and test result informational messages are highlighted in different colors.

Figure 18.5. Output Tab
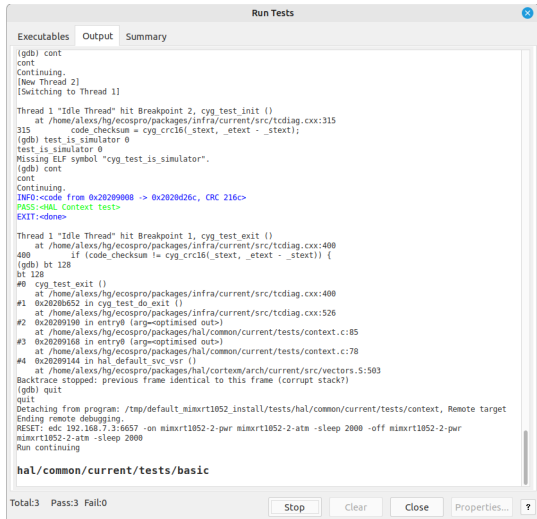
The output text window will follow new output as it is appended to the text window, and the test result totals in the status bar will be updated as tests are completed. When all the tests have completed execution, or execution of the tests is halted through the *Stop* button, the test result totals are appended to the output.

Summary Tab

This tab is used to display a record, in summary form, of those tests executed and is illustrated in Figure 18.6, "Summary Tab".

Figure 18.6. Summary Tab

For each execution, the following information is displayed:

- *Time* - the date and time of execution

- *Host* - the host name of the machine from which the test was downloaded

- *Platform* - the platform on which the test was executed

- *Executable* - the executable (file name) of the test executed

- *Status* - the result of executing the test. This will be one of the following:

  - Not started

  - No result

  - Inapplicable

  - Pass

  - DTimeout

  - Timeout

  - Cancelled

  - Fail

  - Assert fail

- *Size* - the size [stripped/unstripped] of the test executed

- *Download* - the download time [mm:ss/mm:ss] used. The first of the two times displayed represents the actual time used: the second the limit time.

- *Elapsed* - the elapsed time [mm:ss] used.

- *Execution* - the execution time [mm:ss/mm:ss] used. The first of the two times displayed represents the actual time used: the second the limit time.

Double-clicking on a test result, or selecting the test result and pressing **ENTER**, will result in the dialog switching to the *Output* tab and position the output text to the start of that test.

Selecting a header will result in the test results being sorted in ascending order by the field values in that column.

Note

New test results will always appear at the bottom of the list, causing the list to be out of order after it has been sorted. Resorting the list will restore the order.

Chapter 19. Creating a Shell

To call up a shell window, select Tools → Shell. Under Windows, you will get a Command Shell with eCosPro 3.1 and later releases, or a Cygwin shell similar to the one below for earlier releases. On Linux, you will get a standard terminal window.

Figure 19.1. Bash Shell
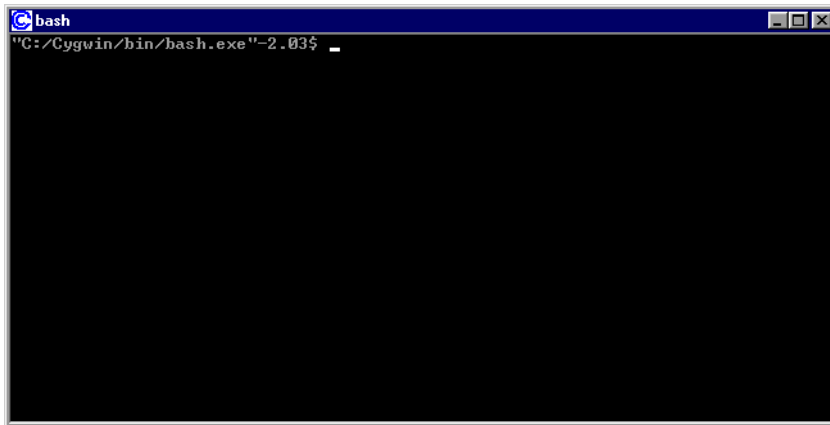
Chapter 20. Keyboard Accelerators

The following table presents the list of keyboard accelerators that can be used with the Configuration Tool.

Table 20.1. Keyboard accelerators

Accelerator	Action	Remarks
Alt+**1**	*hide/show properties window*	
Alt+**2**	*hide/show output window*	
Alt+**3**	*hide/show short description window*	
Alt+**4**	*hide/show conflicts window*	
Ctrl+**A**	*select all*	output window and in-cell editing
Ctrl+**C**	*copy*	output window and in-cell editing
Ctrl+**F**	Edit → Find	
Ctrl+**N**	File → New (`target:template`)	
Alt+**N**	File → New Target	
Ctrl+**O**	File → Open	
Ctrl+**S**	File → Save	
Ctrl+**V**	*Paste*	in-cell editing only
Ctrl+**X**	*Cut*	in-cell-editing only
Ctrl+**Z**	*Undo*	in-cell editing only
F1	*Context-sensitive help*	
F3	*Find next*	
F7	Build → Library	
Shift+**F7**	Build → Tests	
Alt+**F6**	View → Next window	
Shift+**Alt**+**0**	View → Previous window	
Shift+**Ins**	*Paste*	in-cell editing only
Shift+**F10**	*Display context menu*	Configuration window
Alt+**Enter**	*Display properties dialog box*	Configuration window
>	*Increment item value*	Configuration window
<	*Decrement item value*	Configuration window
Space	*Toggle item value*	Configuration window
Ctrl+**I**	Help → Repository Information	

Chapter 21. Checking for updates

eCosCentric periodically releases updates to eCosPro Developer's Kits, GNU Toolchains, the eCosPro Host Tools and Eclipse. At start up, the configuration tool will automatically check the eCosPro Support Portal for updates to the installed packages. If an update is detected, the dialog window illustrated below will appear. Detection of available updates requires that the host machine has a direct access to the internet. Proxies are currently not supported. The update check feature was introduced in eCosPro 4.0.

Figure 21.1. eCosPro Updates Available
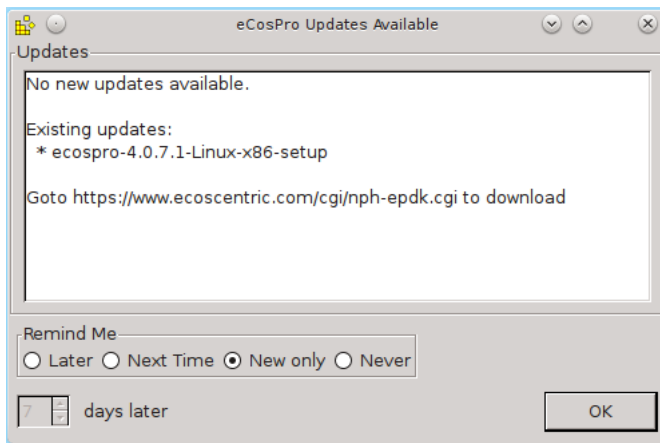
This check may also be performed manually from the configuration tool menu option Tools → Check for updates. New updates and any updates already reported are listed in the `Updates` area of the dialog. The developer may also change how they are reminded (periodically, new updates, or never) as follows:

- *Later* - the developer will only be reminded again of updates not yet installed when the configuration tool is next run, after the specified number of days. The developer is notified immediately of any new updates, irrespective of this setting.

- *Next Time* - the developer will be reminded of any updates not yet installed, and any new updates available, the next time the configuration tool is run.

- *New only* - the developer will be informed only of new updates the next time the configuration tool is run. No reminders will be given for updates for which the developer has already been notified.

- *Never* - the developer will never be informed of new updates and never be reminded of existing updates not yet installed.

  Manual checks for updates can only be performed through the configuration tool menu option Tools → Check for updates.

Part IV. eCos Programming Concepts and Techniques

Programming with eCos is somewhat different from programming in more traditional environments. eCos is a configurable open source system, and you are able to configure and build a system specifically to meet the needs of your application.

Various different directory hierarchies are involved in configuring and building the system: the *component repository*, the *build tree*, and the *install tree*. These directories exist in addition to the ones used to develop applications.

Table of Contents

Chapter 22. CDL Concepts

About this chapter

This chapter serves as a brief introduction to the concepts involved in eCos (Embedded Configurable Operating System). It describes the configuration architecture and the underlying technology to a level required for the embedded systems developer to configure eCos. It does not describe in detail aspects such as how to write reusable components for eCos: this information is given in the *Component Writer,s Guide*.

Background

Software solutions for the embedded space place particularly stringent demands on the developer, typically represented as requirements for small memory footprint, high performance and robustness. These demands are addressed in eCos by providing the ability to perform compile-time specialization: the developer can tailor the operating system to suit the needs of the application. In order to make this process manageable, eCos is built in the context of a Configuration Infrastructure: a set of tools including a Configuration Tool and a formal description of the process of configuration by means of a *Component Definition Language*.

Configurations

eCos is tailored at source level (that is, before compilation or assembly) in order to create an eCos *configuration*. In concrete terms, an eCos configuration takes the form of a configuration save file (with extension .ecc) and set of files used to build user applications (including, when built, a library file against which the application is linked).

Component Repository

eCos is shipped in source in the form of a *component repository* - a directory hierarchy that contains the sources and other files which are used to build a configuration. The component repository can be added to by, for example, downloading from the net.

Component Definition Language

Part of the component repository is a set of files containing a definition of its structure. The form used for this purpose is the *Component Definition Language* (CDL). CDL defines the relationships between components and other information used by tools such as the eCos Configuration Tool. CDL is generally formulated by the writers of components: it is not necessary to write or understand CDL in order for the embedded systems developer to construct an eCos configuration.

Packages

The building blocks of an eCos configuration are called *packages*. Packages are the units of software distribution. A set of core packages (such as kernel, C library and math library) additional packages provided by eCosCentric (such as C++ library and USB device support): additional third-party packages will be available in future.

A package may exist in one of a number of *versions*. The default version is the *current* version. Only one version of a given package may be present in the component repository at any given time.

Packages are organized in a tree hierarchy. Each package is either at the top-level or is the child of another package.

The eCos Package Administration Tool can be used to add or remove packages from the component repository. The eCos Configuration Tool can be used to include or exclude packages from the configuration being built.

Configuration Items

Configuration items are the individual entities that form a configuration. Each item corresponds to the setting of a C pre-processor macro (for example, `CYGHWR_HAL_ARM_PID_GDB_BAUD`). The code of eCos itself is written to test such pre-processor macros so as to tailor the code. User code can do likewise.

Configuration items come in the following flavors:

- *None*: such entities serve only as place holders in the hierarchy, allowing other entities to be grouped more easily.

- *Boolean* entities are the most common flavor; they correspond to units of functionality that can be either enabled or disabled. If the entity is enabled then there will be a #define; code will check the setting using, for example, #ifdef

- *Data* entities encapsulate some arbitrary data. Other properties such as a set or range of legal values can be used to constrain the actual values, for example to an integer or floating point value within a certain range.

- *Booldata* entities combine the attributes of *Boolean* and *Data*: they can be enabled or disabled and, if enabled, will hold a data value.

Like packages, configuration items exist in a tree-based hierarchy: each configuration item has a parent which may be another configuration item or a package. Under some conditions (such as when packages are added or removed from a configuration), items may be "re-parented" such that their position in the tree changes.

Expressions

Expressions are relationships between CDL items. There are three types of expression in CDL:

Table 22.1. CDL Expressions

Expression Type	Result	Common Use (see Table 22.2, "Configuration properties")
Ordinary	A single value	legal_values property
List	A range of values (for example "1 to 10")	legal_values property
Goal	True or False	requires and active_if properties

Properties

Each configuration item has a set of properties. The following table describes the most commonly used:

Table 22.2. Configuration properties

Property	*Use*
Flavor	The "type" of the item, as described above
Enabled	Whether the item is enabled
Current_value	The current value of the item
Default_value	An ordinary expression defining the default value of the item
Legal_values	A list expression defining the values the item may hold (for example, 1 to10)
Active_if	A goal expression denoting the requirement for this item to be active (see below: *Inactive Items*)

Property	Use
Requires	A goal expression denoting requirements this item places on others (see below: *Conflicts*)
Calculated	Whether the item as non-modifiable
Macro	The corresponding C pre-processor macro
File	The C header file in which the macro is defined
URL	The URL of a documentation page describing the item
Hardware	Indicates that a particular package is related to specific hardware

A complete description of properties is contained in the *Component Writer,s Guide*.

Inactive Items

Descendants of an item that is disabled are inactive: their values may not be changed. Items may also become *inactive* if an active_if expression is used to make the item dependent on an expression involving other items.

Conflicts

Not all settings of configuration items will lead to a coherent configuration; for example, the use of a timeout facility might require the existence of timer support, so if the one is required the other cannot be removed. Coherence is policed by means of consistency rules (in particular, the goal expressions that appear as CDL items *requires* and *active_if* attributes [see above]). A violation of consistency rules creates a *conflict*, which must be resolved in order to ensure a consistent configuration. Conflict resolution can be performed manually or with the assistance of the eCos tools. Conflicts come in the following flavors:

- An *unresolved* conflict means that there is a reference to an entity that is not yet in the current configuration

- An *illegal value* conflict is caused when a configuration item is set to a value that is not permitted (that is, a *legal_values* goal expression is failing)

- An *evaluation exception* conflict is caused when the evaluation of an expression would fail (for example, because of a division by zero)

- An *unsatisfied goal* conflict is caused by a failing *requires* goal expression

- A *bad data* conflict arises only rarely, and corresponds to badly constructed CDL. Such a conflict can only be resolved by reference to the CDL writer.

Templates

A *template* is a saved configuration - that is, a set of packages and configuration item settings. Templates are provided with eCos to allow you to get started quickly by instantiating (copying) a saved configuration corresponding to one of a number of common scenarios; for example, a basic eCos configuration template is supplied that contains the infrastructure, kernel, C and math libraries, plus their support packages.

Chapter 23. The Component Repository and Working Directories

Each of the file trees involved in eCos development has a different role.

Component Repository

The eCos *component repository* contains directories for all the packages that are shipped with eCos or provided by third parties.

The component repository should not be modified as part of application development.

Figure 23.1. Component repository

```
eCos
  doc
    guides
      gnupro-guides
      user-guides
    ref
      ecos-ref
      gnupro-ref
    tutorials
      ecos-tutorial
  ecos-98r1p6
  examples
  loaders
    arm-aeb
    arm-pid
    mn10300-stdeval1
    powerpc-cogent
    sparclite-sleb
    tx39-jmr3904
  packages
    compat
      uitron
    cygmon
      v1_3
    devs
      wallclock
      watchdog
    error
      v1_3
    hal
      arm
      common
      i386
      mips
      mn10300
      powerpc
      sh
      sparclite
    infra
      v1_3
    io
      common
      pci
      serial
    kernel
      v1_3
    language
      c
    pkgconf
    templates
  prebuilt
    sleb
  tools
```

Purpose

The component repository is the master copy of source code for all system and third party components. It also contains some files needed to administer and build the system, such as **ecosadmin.tcl**.

How is it modified?

You modify it by importing new versions of packages from a distribution or removing existing packages. These activities are undertaken using the eCos Package Administration Tool.

When is it edited manually?

Files in the component repository should only be edited manually as determined by the component maintainer.

User Applications

User application source code should *not* go into the component repository.

Examples of files in this hierarchy:

BASE_DIR/doc/ref/ecos-ref.html

> The top level HTML file for the *eCos Reference Manual*.

BASE_DIR/prebuilt/pid/tests/kernel/*<version>*/tests/thread_gdb.exe

BASE_DIR/prebuilt/linux/tests/kernel/*<version>*/tests/thread_gdb.exe

> Pre-built tests for the supported platforms, and the synthetic Linux target.

BASE_DIR/examples/twothreads.c

> One of the example programs.

BASE_DIR/ecosadmin.tcl

> The Tcl program which is used to import new versions of packages from a distribution or remove existing packages.

BASE_DIR/packages/language/c/libm/*<version>*/src/double/portable-api/s_tanh.c

> Implementation of the hyperbolic tangent function in the standard math library.

BASE_DIR/pkgconf/rules.mak

> A file with **make** rules, used by the makefile.

Build Tree

The *build tree* is the directory hierarchy in which all *generated* files are placed. Generated files consist of the makefile, the compiled object files, and a dependency file (with a .d extension) for each source file.

Purpose

The build tree is where all intermediate object files are placed.

How is it modified?

Recompiling can modify the object files.

User applications

User application source or binary code should *not* go in the build tree.

Examples of files in this hierarchy

```
ecos-work/language/c/libc/<version>/src
```

The directory in which object files for the C library are built.

Install Tree

The *install tree* is the location for all files needed for application development. The `libtarget.a` library, which contains the custom-built eCos kernel and other components, is placed in the install tree, along with all packages, public header files. If you build the tests, the test executable programs will also be placed in the install tree.

By default, the install tree is created by **ecosconfig** in a subdirectory of the build tree called `install`. This can be modified with the `--prefix` option (see Chapter 26, *Manual Configuration*).

Purpose

The install tree is where the custom-built `libtarget.a` library, which contains the eCos kernel and other components, is located. The install tree is also the location for all the header files that are part of a published interface for their component.

How is it modified?

Recompiling can replace `libtarget.a` and the test executables.

When is it edited manually?

Where a memory layout requires modification without use of the eCos Configuration Tool, the memory layout files must be edited directly in the install tree. These files are located at `install/include/pkgconf/mlt_*.*`. Note that subsequent modification of the install tree using the Configuration Tool will result in such manual edits being lost.

User applications

User application source or binary code should *not* go in the install tree.

Examples of files in this hierarchy

```
install/lib/libtarget.a
```

The library containing the kernel and other components.

```
install/include/cyg/kernel/kapi.h
```

The header file for the kernel C language API.

`install/include/pkgconf/mlt_arm_pid_ram.ldi`

The linker script fragment describing the memory layout for linking applications intended for execution on an ARM PID development board using RAM startup.

`install/include/stdio.h`

The C library header file for standard I/O.

Application Build Tree

This tree is not part of eCos itself: it is the directory in which eCos end users write their own applications.

Example applications and their `Makefile` are located in the component repository, in the directory `BASE_DIR` /examples.

There is no imposed format on this directory, but there are certain compiler and linker flags that must be used to compile an eCos application. The basic set of flags is shown in the example `Makefile`, and additional details can be found in Chapter 24, *Compiler and Linker Options*.

Chapter 24. Compiler and Linker Options

eCos is built using the GNU C and C++ compilers. eCos relies on certain features of these tools such as constructor priority ordering and selective linking which are not part of other toolchains.

Some GCC options are required for eCos, and others can be useful. This chapter gives a brief description of the required options as well as some recommended eCos-specific options. All other GCC options (described in the GCC manuals) are available.

Compiling a C Application

The following command lines demonstrate the *minimum* set of options required to compile and link an eCos program written in C.

> **Note**
>
> Remember that when this manual shows *TARGET*-**gcc** you should use the full name of the cross compiler, e.g. **i386-elf-gcc**, **arm-eabi-gcc**, or **sh-elf-gcc**. When compiling for the synthetic Linux target, use the native **gcc** which must have the features required by eCos.

```
$ TARGET-gcc -c  -IECOS_INSTALL_DIR/include file.c
$ TARGET-gcc -o program file.o -LECOS_INSTALL_DIR/lib -Ttarget.ld -nostdlib
```

> **Note**
>
> Certain targets may require extra options, for example the SPARClite architectures require the option -mcpu=s-parclite. Examine the *BASE_DIR*/examples/Makefile or the "Global compiler flags" option (CYG-BLD_GLOBAL_CFLAGS) in your generated eCos configuration) to see if any extra options are required, and if so, what they are.
>
> The following command lines use some other options which are recommended because they use the selective linking feature:

```
$ TARGET-gcc -c  -IECOS_INSTALL_DIR/include -I. -ffunction-sections -fdata-sections -g \
  -O2 file.c
$ TARGET-gcc -o program file.o -ffunction-sections -fdata-sections -Wl,--gc-sections -g \
  -O2 -LECOS_INSTALL_DIR/lib -Ttarget.ld -nostdlib
```

Compiling a C++ Application

The following command lines demonstrate the *minimum* set of options required to compile and link an eCos program written in C++.

> **Note**
>
> Remember that when this manual shows *TARGET*-**g++** you should use the full name of the cross compiler, e.g. **i386-elf-g++**, **arm-eabi-g++**, or **sh-elf-g++**. When compiling for the synthetic Linux target, use the native **g++** which must have the features required by eCos.

```
$ TARGET-g++ -c  -IECOS_INSTALL_DIR/include -fno-rtti -fno-exceptions file.cxx
$ TARGET-g++ -o program file.o -LECOS_INSTALL_DIR/lib -Ttarget.ld -nostdlib
```

> **Note**
>
> Certain targets may require extra options, for example the SPARClite architectures require the option -mcpu=s-parclite. Examine the *BASE_DIR*/packages/targets file or *BASE_DIR*/examples/Makefile or

the "Global compiler flags" option (CYGBLD_GLOBAL_CFLAGS) in your generated eCos configuration) to see if any extra options are required, and if so, what they are.

The following command lines use some other options which are recommended because they use the *selective linking* feature:

```
$ TARGET-g++ -c -IECOS_INSTALL_DIR/include -I. -ffunction-sections -fdata-sections -fno-rtti \
  -fno-exceptions -finit-priority -g -O2 file.cxx
$ TARGET-g++ -o program file.o -Wl,--gc-sections -g -O2 -LECOS_INSTALL_DIR/lib \
  -Ttarget.ld -nostdlib
```

Chapter 25. Debugging Techniques

eCos applications and components can be debugged in traditional ways, with printing statements and debugger single-stepping, but there are situations in which these techniques cannot be used. One example of this is when a program is getting data at a high rate from a real-time source, and cannot be slowed down or interrupted.

eCos,s infrastructure module provides a *tracing* formalism, allowing the kernel,s tracing macros to be configured in many useful ways. eCos,s kernel provides *instrumentation buffers* which also collect specific (configurable) data about the system,s history and performance.

Tracing

To use eCos,s tracing facilities you must first configure your system to use *tracing*. You should enable the Asserts and Tracing component (CYGPKG_INFRA_DEBUG) and the Use tracing component within it (CYGDBG_USE_TRACING). These options can be enabled with the Configuration Tool or by editing the file *BUILD_DIR*/pkgconf/infra.h manually.

You should then examine all the tracing-related options in the *Package: Infrastructure* chapter of the *eCos Reference Manual*. One useful set of configuration options are: CYGDBG_INFRA_DEBUG_FUNCTION_REPORTS and CYGDBG_INFRA_DE‐BUG_TRACE_MESSAGE, which are both enabled by default when tracing is enabled.

The following "Hello world with tracing" shows the output from running the hello world program (from) that was built with tracing enabled:

Example 25.1. Hello world with tracing

```
$ mips-tx39-elf-run --board=jmr3904  hello
Hello, eCos world!
ASSERT FAIL: <2>cyg_trac.h  [623] Cyg_TraceFunction_Report_::set_exitvoid()  exitvoid used in typed function
TRACE: <1>mlqueue.cxx       [395] Cyg_ThreadQueue_Implementation::enqueue()  {{enter
TRACE: <1>mlqueue.cxx       [395] Cyg_ThreadQueue_Implementation::enqueue()  }}RETURNING UNSET!
TRACE: <1>mlqueue.cxx       [126] Cyg_Scheduler_Implementation::add_thread() }}RETURNING UNSET!
TRACE: <1>thread.cxx        [654] Cyg_Thread::resume()                       }}return void
TRACE: <1>cstartup.cxx      [160] cyg_iso_c_start()                          }}return void
TRACE: <1>startup.cxx       [142] cyg_package_start()                        }}return void
TRACE: <1>startup.cxx       [150] cyg_user_start()                           {{enter
TRACE: <1>startup.cxx       [150] cyg_user_start()                           (((void)))
TRACE: <1>startup.cxx       [153] cyg_user_start()        'This is the system default cyg_user_start()'
TRACE: <1>startup.cxx       [157] cyg_user_start()                           }}return void
TRACE: <1>sched.cxx         [212] Cyg_Scheduler::start()                     {{enter
TRACE: <1>mlqueue.cxx       [102] Cyg_Scheduler_Implementation::schedule()   {{enter
TRACE: <1>mlqueue.cxx       [437] Cyg_ThreadQueue_Implementation::highpri()  {{enter
TRACE: <1>mlqueue.cxx       [437] Cyg_ThreadQueue_Implementation::highpri()  }}RETURNING UNSET!
TRACE: <1>mlqueue.cxx       [102] Cyg_Scheduler_Implementation::schedule()   }}RETURNING UNSET!
TRACE: <2>intr.cxx          [450] Cyg_Interrupt::enable_interrupts()         {{enter
TRACE: <2>intr.cxx          [450] Cyg_Interrupt::enable_interrupts()         }}RETURNING UNSET!
TRACE: <2>thread.cxx        [ 69] Cyg_HardwareThread::thread_entry()         {{enter
TRACE: <2>cstartup.cxx      [127] invoke_main()                             {{enter
TRACE: <2>cstartup.cxx      [127] invoke_main()                             ((argument is ignored))
TRACE: <2>dummyxxmain.cxx   [ 60] __main()                                  {{enter
TRACE: <2>dummyxxmain.cxx   [ 60] __main()                                  (((void)))
TRACE: <2>dummyxxmain.cxx   [ 63] __main()                'This is the system default __main()'
TRACE: <2>dummyxxmain.cxx   [ 67] __main()                                  }}return void
TRACE: <2>memcpy.c          [112] _memcpy()                                 {{enter
TRACE: <2>memcpy.c          [112] _memcpy()                                 ((dst=80002804, src=BFC14E58, n=19))
TRACE: <2>memcpy.c          [164] _memcpy()                                 }}returning 80002804
TRACE: <2>cstartup.cxx      [137] invoke_main()          'main() has returned with code 0. Calling exit()'
TRACE: <2>exit.cxx          [ 71] __libc_exit()                             {{enter
TRACE: <2>exit.cxx          [ 71] __libc_exit()                             ((status=0 ))
TRACE: <2>atexit.cxx        [ 84] cyg_libc_invoke_atexit_handlers()         {{enter
TRACE: <2>atexit.cxx        [ 84] cyg_libc_invoke_atexit_handlers()         (((void)))
Scheduler:
Lock:                0
Current Thread:      <null>
Threads:
Idle Thread          pri =  31 state = R      id =   1
stack base = 800021F0 ptr = 80002510 size = 00000400
sleep reason NONE     wake reason NONE
```

```
queue = 80000C54      wait info = 00000000
<null>                pri =   0 state = R     id =   2
stack base = 80002A48 ptr = 8000A968 size = 00008000
sleep reason NONE     wake reason NONE
queue = 80000BD8      wait info = 00000000
```

Instrumentation

Instrumentation is a valuable performance analysis and debugging mechanism that can be useful when porting eCos, developing drivers and other subsystems, and during eCos application development. It can provide insights into the real-time behaviour of the system - encompassing the ordering, frequency and timing of selected system and application events, whilst also providing associated event specific context information.

Instrumentation works by selecting the set of events of interest during configuration, followed by the runtime capture and logging of these events into either a in-memory buffer for later extraction, or immediately via an external hardware debug interface. The resulting binary instrumentation log is then processed by a host-based tool to either convert into a human-readable textual format, or into formats supported by various graphical trace viewers.

Examples of the use of instrumentation include analysis of the timing and ordering of control flow between different threads, the duration of interrupt service routines, or within a specific device driver to identify transitions between various states of its state machine.

The run-time instrumentation support is provided by the the `CYGPKG_KERNEL` package, but is not limited to supporting the Kernel. Instrumentation *event* points can be added to packages or user applications as required.

The Kernel header file `cyg/kernel/instrmnt.h` provides the implementation definitions, and should normally be included by any code wishing to use instrumentation.

The fundamental identity of instrumentation points is the event "type" value, which comprises an 8-bit `class` value and a 5-bit `code` value (range 1..31). The value 0 is reserved in both the class and code space. Some further type bits define how many argument values are associated to an event type. The class space is further sub-divided into space for the Kernel instrumentation (previously the only classes defined and supported), space for well-known (standard eCos) packages and space from `CYG_INSTRU-MENT_CLASS_APPLICATION_BASE` set aside for per-application tracing.

> ### Note
>
> It is the responsibility of the developer to manage the use of the instrumentation application space within their own code. The currently defined application class/code space allows for `1953` unique instrumentation identifiers.

`CYGDBG_KERNEL_INSTRUMENT_FLAGS`

The configuration option `CYGDBG_KERNEL_INSTRUMENT_FLAGS` controls whether run-time support for enabling/disabling individual event class/code output is provided. If the feature is disabled then *ALL* instrumentation macro calls will result in instrumentation records being generated. The sub-option `CYGDBG_KERNEL_INSTRUMENT_FLAGS_ALL` controls whether the run-time flag support is only provided for the original Kernel instrumentation classes (disabled), or for the complete event class range (enabled). Normally this option would be disabled since supporting run-time flags for the complete class range has a large memory footprint, and if disabled non-Kernel packages can implement their own run-time control if needed.

`CYGPKG_KERNEL_INSTRUMENT_BUFFER`

The configuration option `CYGPKG_KERNEL_INSTRUMENT_BUFFER` controls whether the default Kernel memory buffer based instrumentation is used, or an alternative instrumentation system is being provided by other packages or the application run-time. Certain architectures, or specific target platforms, may provide alternative implementations that provide for direct low-overhead output of instrumentation. For example, some Cortex-M designs may include support for outputting instrumentation immediately via the on-chip ITM stimulus port interfaces to a suitable host SWD tool, allowing for continuous streams of instrumentation events to be captured.

`CYGPKG_KERNEL_INSTRUMENT_TIMESTAMPS`

> The configuration option `CYGPKG_KERNEL_INSTRUMENT_TIMESTAMPS`, along with the sub-configuration items it enables, control whether the run-time includes timestamps in event records. For some applications the delta between events is not important, only the order that events happen in. By disabling timestamps then the memory footprint of individual instrumentation records can be reduced. Alternative direct output implementations might provide their own implicit timestamp capability - in which case the built-in timestamp feature would not be required.

The default implementation makes use of local memory *instrument buffers* to hold a number of event records. Whilst being simple, the use of memory buffers for recording instrumentation events does impose a limitation on the number of events that can be held at any single moment, and the amount of memory available for holding instrumentation records may be severely limited on deeply embedded systems with small amounts of available RAM.

Kernel Instrumentation

The `CYGPKG_KERNEL` package also provides instrumentation covering the major components of the eCos Kernel. Examples of Kernel *events* that can be monitored are:

- scheduler events

- thread operations

- interrupts

- mutex operations

- binary semaphore operations

- counting semaphore operations

- clock ticks and interrupts

Examples of fine-grained scheduler event types are:

- scheduler lock

- scheduler unlock

- rescheduling

- time slicing

Embedded (non-loaded) information

To allow for generic post-processing tools (that are not tied to any specific eCos configuration, architecture or platform) the configuration option `CYGIMP_KERNEL_INSTRUMENT_IMAGE_SECTIONS` can be enabled to provide instrumentation description information that is embedded in the `ELF` application image.

Note

> Even when the binary code and initialised data from an application image are written to ROM, it is important to keep a copy of the original application image for future reference. This is so that any instrumentation data that is generated by executing the application code can be interpreted correctly. Keeping a copy of the application image used for releases is always a good idea anyway, since the image file contains symbolic and debugging information that may also be useful at a later date if problems need to be investigated.

The instrumentation description information is encoded in specifically named sections that are *NOT* referenced from the actual application code or data, and so do not form part of the loaded data or ROM image. The sections are purely present to provide details of the actual instrumentation generated by the application using the specific eCos configuration the application was built against.

For example, an embedded section describing the format of the actual event records may be provided. Such a section will contain information about the field sizes and types so that the external tool can determine what fields were actually used, in which order, with what alignment or endianness.

Similarly the mapping between the internal event record class and code values to the fixed ASCII tags used to identify particular events is provided in an embedded section. This ensures that regardless of the actual binary value used to encode the event in the event record, the external tool used to analyse instrumentation event records can correctly identify (for example) the SCHED:LOCKED event.

The header file `cyg/kernel/instrument_data.h` defines the layout of these embedded structures. Even though the header is provided within the eCos Kernel source tree, it is common amongst all eCos releases and when updated will ensure backwards compatibility so that any external tool built against the header will still be valid and will *NOT* need to be re-built just because a newer eCos release is being used.

The use of such structures embedded in the executable image ensures that the description of instrumentation records produced is held with the actual application image; and there does not need to be magic collusion between the external tools and any specific eCos configuration used for interpreting the generated data. The information embedded is designed to be future-proof to allow "old" tools to extract the information they can process from newer binaries that may actually provide more embedded data.

instdump tool

The Kernel package includes the source code for the **instdump** command line tool. The **instdump** tool can be used to convert binary instrumentation data created by an application into human readable form, or into formats required by specific trace viewers. It can also be used to examine the information embedded in application images. The source for the **instdump** tool can be found in the `BASE_DIR`/packages/kernel/`<version>`/host/instr directory. The tool currently requires `libelf` for parsing the application image to extract the embedded information, and `libiberty` for C++ symbol de-mangling. It can be built as follows on a suitable Linux host:

```
$ cd $ECOS_REPOSITORY/kernel/$ECOS_VERSION/host/instr
$ gcc -o instdump instdump.c -lelf -liberty
```

A Windows binary can be cross-built on a Linux host, assuming a suitable mingw environment is available. The following example build sequence assumes that the relevant `libiberty` and `libelf` sources have been downloaded and un-packed into the relevant `/tmp/mingw` directories.

An example mingw `libelf` cross-build:

```
$ mkdir -p /tmp/mingw/build_libelf
$ cd /tmp/mingw/build_libelf
$ AR="i686-pc-mingw32-ar" CC="i686-pc-mingw32-gcc" ../libelf-0.8.13/configure --disable-shared \
  --disable-nls --enable-compat --target=i686-pc-mingw32
$ make
$ i686-pc-mingw32-ranlib lib/libelf.a
```

An example mingw `libiberty` cross-build:

```
$ mkdir -p /tmp/mingw/build_libiberty
$ cd /tmp/mingw/build_libiberty
$ CC="i686-pc-mingw32-gcc" ../libiberty/configure --host=i686-pc-mingw32 --disable-shared
$ make
```

Once the static libraries are available the actual **instdump.exe** can be compiled and linked by executing the following commands:

```
$ cd $ECOS_REPOSITORY/kernel/$ECOS_VERSION/host/instr
$ i686-pc-mingw32-gcc -I/tmp/mingw/libelf-0.8.13/lib/ -I/tmp/mingw/build_libelf/lib/ \
  -o instdump.exe instdump.c /tmp/mingw/build_libelf/lib/libelf.a \
```

```
    /tmp/mingw/build_libiberty/libiberty/libiberty.a
```

In the future extended versions of **instdump** or additional tools may be provided to make use of the embedded (INFO) sections for specific uses. For example, to convert the eCos specific binary instrumentation into other trace/instrumentation formats such as the Common Trace Format (CTF). The source code of **instdump** provides a useful example of how to extract and process the embedded sections and eCos trace information.

The tool provides a simple overview of its options by using the --help option.

The instdump tool can then be used to display embedded (INFO) data held within an ELF application image for verification of the embedded data. e.g.

```
$ instdump elfimage
```

It can also interpret raw binary instrumentation dump as produced by executing the elfimage application with whatever instrumentation capture support is suitable. e.g.

```
$ instdump --record-format instdata.bin elfimage
```

The --instrumentation option allows a binary file of contiguous records to be passed. This would normally be used when examining memory buffer captures. The --record-format option allows the Cortex-M specific ITM record format to be parsed as captured by suitable SWD hardware debug interfaces (e.g. OpenOCD ST-LINKv2, PEEDI).

Further, when instrumentation data is being parsed then the --chrome-json option can be used to have **instdump** generate JSON trace-viewer compatible output. This can be displayed using the trace viewer that is built in to the Chrome browser. You can activate this by entering about:tracing as the URL, and then clicking the Load button on the tool bar that then appears.

If needed the example tool can extract the binary (INFO) sections if they need to be processed by some other tool. e.g.

```
$ instdump --only-section .ecos_inst_time -o time.bin elfimage
```

Though it should be noted that the standard objcopy tool also provides support for extracting arbitrary sections from an executable.

For backwards compatibility it can also be used to generate an instrument_desc.h compatible header.

```
$ instdump --gen-header --output instrument_desc.h elfimage
```

Adding new instrumentation

At its simplest adding new instrumentation is simply a case of defining manifests for the class, and the (up to 31) sub-codes within that class; and specifying the number of arguments (currently 0, 1 or 2) that the class/code pairing provides.

To make full use of the support provided by the Kernel package instrmnt.h header file, the manifests defining new event classes, and the event codes within each class, should adhere to the relevant CYG_INSTRUMENT_ prefix form. This allows the macros provided by the instrmnt.h header to be used to insert instrumentation points and to declare the mappings in the embedded information structures.

Note

Due to backwards compatibility with the previous instrumentation implementation, which is still supported for the moment, there can be *NO* whitespace in the manifest definitions for the individual event code definitions, e.g. the CYG_INSTRUMENT_EVENT_ prefix. When the previous implementation no longer needs to be supported then this limitation will be lifted. This is due to the use of a simple build-time script to parse the instrmnt.h source file.

Example 25.2. Including instrumentation

The following simple example inserts an instrumentation point between the two function calls. If instrumentation is disabled in the configuration; then no code will be produced for the CYG_INSTRUMENT() macro call.

```
#include <cyg/kernel/instrmnt.h>
// Normally would be in a shared header file for use by the whole application
#define CYG_INSTRUMENT_CLASS_OURCLASS 0xc000
#define CYG_INSTRUMENT_EVENT_OURCLASS_OURCODE (CYG_INSTRUMENT_ARGS_VAL(2)|5)
void some_function(cyg_uint32 arg1,cyg_uint32 arg2) {
  do_something();
  CYG_INSTRUMENT(OURCLASS,OURCODE,arg1,arg2);
  do_more_stuff();
}
```

When the `CYGIMP_KERNEL_INSTRUMENT_IMAGE_SECTIONS` feature is enabled then for each new event class being defined then suitable descriptors should be provided *ONCE* in a `CYG_KERNEL_INSTRUMENT_SECTION_EVENT_DESCRIPTORS` linked section so that the class/code mappings for an event are available to the external processing tools.

Example 25.3. Providing class/code information

The following is a simple example of how the event class/code information is provided in the embedded section for future access by suitable external processing tools. This example provides the definitions for the named "OURCLASS", "EXAMPLE" and "TEST" classes.

```
#if defined(CYGIMP_KERNEL_INSTRUMENT_IMAGE_SECTIONS)
const struct instrument_event_descriptor
ecos_kernel_instrument[] CYGBLD_ATTRIB_SECTION(CYG_KERNEL_INSTRUMENT_SECTION_EVENT_DESCRIPTORS) = {
  // Event class:
  CYG_INSTRUMENT_DESCRIPTOR_ECLASS(OURCLASS),
  // Event code entries:
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(OURCLASS,OURCODE),
  // Event class:
  CYG_INSTRUMENT_DESCRIPTOR_ECLASS(EXAMPLE),
  // Event code entries:
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(EXAMPLE,POINT1),
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(EXAMPLE,SOMEWHERE),
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(EXAMPLE,STARTPOINT),
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(EXAMPLE,ENDPOINT),
  // Event class:
  CYG_INSTRUMENT_DESCRIPTOR_ECLASS(TEST),
  // Event code entries:
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(TEST,ERROR1),
  CYG_INSTRUMENT_DESCRIPTOR_ECODE(TEST,KEYPRESSED),
  // Terminator
  { 0x00, 0x00, 0x00, { "" } },
};
#endif // CYGIMP_KERNEL_INSTRUMENT_IMAGE_SECTIONS
```

The source file `cyg/kernel/instrument_kernel.h` contains a good real-world example. It provides the definitions for the Kernel provided instrumentation classes and the sub-codes for each class.

Memory buffer instrumentation

The default instrumentation implementation (when `CYGPKG_KERNEL_INSTRUMENT_BUFFER` is defined) uses *instrument buffers*.

Information about the events is stored in an *event record*. The structure that defines this record has type struct Instrument_Record. As previously described in the section called "Embedded (non-loaded) information", when `CYGIMP_KERNEL_INSTRUMENT_IMAGE_SECTIONS` is enabled then a description of the memory layout of instrumentation records is embedded into the linked application image. This is so that external tools can process the records written by the application without having to have access to the build source used to create that application, or for the tool to be created against a specific revision of the source used to create the application.

The list of records is stored in an array called instrument_buffer which you can let the kernel provide or you can provide yourself by setting the configuration option `CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER`.

To write a program that examines the instrumentation buffers:

1. Enable instrumentation buffers in the eCos kernel configuration. The component macro is `CYGPKG_KERNEL_INSTRUMENT`.

2. To allocate the buffers yourself, enable the configuration option `CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER`.

3. Include the header file `cyg/kernel/instrmnt.h`:

```
#include <cyg/kernel/instrmnt.h>
```

4. Enable the events you want to record using `cyg_instrument_enable()`, and disable them later. Look at `cyg/kernel/instrmnt.h` and the examples below to see what Kernel events can be enabled.

5. Place the code you want to debug between the matching functions `cyg_instrument_enable()` and `cyg_instrument_disable()`.

6. Examine the buffer either from within the application, from an attached debugger (e.g. GDB), or from an external (post-processing) tool with a binary image of the captured instrumentation.

Instrumentation buffer API

If the CDL option `CYGDBG_KERNEL_INSTRUMENT_BUFFER_API` is enabled then a simple API is provided to allow for the application to cleanly access the instrumentation buffer itself. Without this API the application is itself responsible for ensuring serialised access to the instrumentation memory to ensure that valid/complete records are interpreted.

> **Note**
>
> This API may be extended in the future to allow for multiple instrumentation buffers to be supported, allowing for better memory buffer instrumentation coverage (to minimise the times when records may be missed due to unavailability of buffer space).

The API has the following interface functions:

`void *cyg_instrument_buffer_claim(size_t *bsize)`

For the current single buffer implementation this just claims the instrumention lock for the duration that the caller holds the buffer. Any instrumentation events raised during the client processing will be dropped.

`void cyg_instrument_buffer_release(void *ibuff)`

Return a claimed buffer back to the instrumentation world. For the current single instrumentation buffer implementation this just releases the held lock.

`size_t cyg_instrument_buffer_size(void)`

This function returns the number of bytes used in the `instrument_buffer`. This is most useful when an external buffer is being provided that is not an exact multiple of the `Instrument_Record` structure being written.

If needed the internal buffer lock control can be directly accessed via the `cyg_instrument_lock()` and `cyg_instrument_unlock()` functions. When the lock is held any calls to the underlying `cyg_instrument()` function will result in *missed* events since the aim is to never block the run-time processing from within the instrumentation recording support.

Using GDB to save instrument buffers

If GDB is being used to control the execution of the application, then when the CPU is halted it is very easy to save the instrumentation buffer to a file on the host. From the GDB prompt we can do something like:

```
(gdb) dump binary memory test.bin &instrument_buffer[0] &instrument_buffer[256]
```

to get hold of the binary data. For a system configured with buffer wrapping capture enabled (CYGDBG_KERNEL_INSTRUMEN-T_BUFFER_WRAP) this does not track where the active "head" of the captured data is. For "halt when full" capture then the above command is sufficient to record the instrumentation data.

When wrapping support is being used then the pointer variable instrument_buffer_pointer references the next slot to be filled. In conjunction with information regarding the instrument_buffer:

```
(gdb) p sizeof(instrument_buffer)
```

we can just access the relevant parts of the instrument_buffer address space and save the image in two parts if we want a time-ordered capture:

```
(gdb) dump binary memory file
  start
  end

(gdb) append binary memory file
  start
  end
```

It is envisaged that generic host tools like instdump (as described in the section called " instdump tool") will be used to examine the instrumentation data produced when used in conjunction with the ELF image for the application that generated the binary data.

Simple application access

Example 25.4. Using instrument buffers

This program is also provided in the examples directory, and is a very simple example of directly decoding the instrumentation buffer from within the application.

```
/* this is a program which uses eCos instrumentation buffers; it needs
  to be linked with a kernel which was compiled with support for
  instrumentation */
#include <stdio.h>
#include <pkgconf/kernel.h>
#include <cyg/kernel/instrmnt.h>
#include <cyg/kernel/kapi.h>
#ifndef CYGPKG_KERNEL_INSTRUMENT_BUFFER
# error You must configure eCos with CYGPKG_KERNEL_INSTRUMENT_BUFFER
#endif
#ifndef CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER
# error You must configure eCos with CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER
#endif
struct Instrument_Record instrument_buffer[20];
cyg_uint32 instrument_buffer_size = 20;
int main(void)
{
  int i;
  cyg_instrument_enable(CYG_INSTRUMENT_CLASS_CLOCK, 0);
  cyg_instrument_enable(CYG_INSTRUMENT_CLASS_THREAD, 0);
  cyg_instrument_enable(CYG_INSTRUMENT_CLASS_ALARM, 0);
  printf("Program to play with instrumentation buffer\n");
  cyg_thread_delay(2);
  cyg_instrument_disable(CYG_INSTRUMENT_CLASS_CLOCK, 0);
  cyg_instrument_disable(CYG_INSTRUMENT_CLASS_THREAD, 0);
  cyg_instrument_disable(CYG_INSTRUMENT_CLASS_ALARM, 0);
  for (i = 0; i < instrument_buffer_size; ++i) {
    printf("Record %02d: type 0x%04x, thread %d, ",i, instrument_buffer[i].type, instrument_buffer[i].thread);
#if defined(CYGPKG_KERNEL_INSTRUMENT_TIMESTAMPS_HAL)
    printf("time %5d, ", instrument_buffer[i].timestamp_hal);
#endif // CYGPKG_KERNEL_INSTRUMENT_TIMESTAMPS_HAL
    printf("arg1 0x%08x, arg2 0x%08x\n", instrument_buffer[i].arg1, instrument_buffer[i].arg2);
```

```
  }
  return 0;
}
```

Here is how you could compile and run this program in the `examples` directory, using (for example) the MN10300 simulator target:

```
$ make XCC=mn10300-elf-gcc ECOS_INSTALL_DIR=/tmp/ecos-work-mn10300/install instrument-test
  mn10300-elf-gcc -c -o instrument-test.o -g -Wall -I/tmp/ecos-work-mn10300/install/include \
    -ffunction-sections -fdata-sections instrument-test.c
  mn10300-elf-gcc -nostartfiles -L/tmp/ecos-work-mn10300/install/lib -W1,--gc-sections -o \
    instrument-test instrument-test.o -Ttarget.ld -nostdlib
$ mn10300-elf-run --board=stdeval1 instrument-test
```

Example 25.5. Instrument buffer output

Here is the output of the **instrument-test** program. Notice that in little over 2 seconds, and with very little activity, and with few event types enabled, it gathered 17 records. In larger programs it will be necessary to select very few event types for debugging.

```
Program to play with instrumentation buffer
Record 00: type 0x0207, thread 2, time  6057, arg1 0x48001cd8, arg2 0x00000002
Record 01: type 0x0202, thread 2, time  6153, arg1 0x48001cd8, arg2 0x00000000
Record 02: type 0x0904, thread 2, time  6358, arg1 0x48001d24, arg2 0x00000000
Record 03: type 0x0905, thread 2, time  6424, arg1 0x00000002, arg2 0x00000000
Record 04: type 0x0906, thread 2, time  6490, arg1 0x00000000, arg2 0x00000000
Record 05: type 0x0901, thread 2, time  6608, arg1 0x48009d74, arg2 0x48001d24
Record 06: type 0x0201, thread 2, time  6804, arg1 0x48001cd8, arg2 0x480013e0
Record 07: type 0x0803, thread 1, time    94, arg1 0x00000000, arg2 0x00000000
Record 08: type 0x0801, thread 1, time   361, arg1 0x00000000, arg2 0x00000000
Record 09: type 0x0802, thread 1, time   548, arg1 0x00000001, arg2 0x00000000
Record 10: type 0x0803, thread 1, time    94, arg1 0x00000000, arg2 0x00000000
Record 11: type 0x0801, thread 1, time   361, arg1 0x00000001, arg2 0x00000000
Record 12: type 0x0903, thread 1, time   513, arg1 0x48009d74, arg2 0x48001d24
Record 13: type 0x0208, thread 1, time   588, arg1 0x00000000, arg2 0x00000000
Record 14: type 0x0203, thread 1, time   697, arg1 0x48001cd8, arg2 0x480013e0
Record 15: type 0x0802, thread 1, time   946, arg1 0x00000002, arg2 0x00000000
Record 16: type 0x0201, thread 1, time  1083, arg1 0x480013e0, arg2 0x48001cd8
Record 17: type 0x0000, thread 0, time     0, arg1 0x00000000, arg2 0x00000000
Record 18: type 0x0000, thread 0, time     0, arg1 0x00000000, arg2 0x00000000
Record 19: type 0x0000, thread 0, time     0, arg1 0x00000000, arg2 0x00000000
```

Part V. Configuration and the Package Repository

The following chapters contain information on running **ecosconfig** (the command line tool that manipulates configurations and constructs build trees) and on managing a source repository across multiple versions of eCos.

Table of Contents

Chapter 26. Manual Configuration

eCos developers will generally use the graphical Configuration Tool for configuring an eCos system and building the target library. However, some user prefer to use command line tools. These command line tools can also be used for batch operations on all platforms, for example as part of a nightly rebuild and testing procedure.

In the current release of the system the command line tools do not provide exactly the same functionality as the graphical tool. Most importantly, there is no facility to resolve configuration conflicts interactively.

The eCos configuration system, both graphical and command line tools, are under constant development and enhancement. Developers should note that the procedures described may change considerably in future releases.

Directory Tree Structure

When building eCos there are three main directory trees to consider: the source tree, the build tree, and the install tree.

The source tree, also known as the component repository, is read- only. It is possible to use a single component repository for any number of different configurations, and it is also possible to share a component repository between multiple users by putting it on a network drive.

The build tree contains everything that is specific to a particular configuration, including header and other files that contain configuration data, and the object files that result from compiling the system sources for this configuration.

The install tree is usually located in the `install` subdirectory of the build tree. Once an eCos system has been built, the install tree contains all the files needed for application development including the header files and the target library. By making copies of the install tree after a build it is possible to separate application development and system configuration, which may be desirable for some organizations.

Creating the Build Tree

Generating a build tree is a non-trivial operation and should not be attempted manually. Instead, eCos is shipped with a tool called **ecosconfig** that should be used to create a build tree.

Usually **ecosconfig** will be run inside the build tree itself. If you are creating a new build tree then typically you will create a new empty directory using the **mkdir** command, **cd** into that directory, and then invoke **ecosconfig** to create a configuration. Alternatively you can also use the `--builddir` option to specify the build directory if the build directory is to be stored in a location other than the current directory. By default, the configuration is stored in a file `ecos.ecc` in the current directory. The configuration may be modified by editing this file directly. **ecosconfig** itself deals with a number of coarse- grained configuration options such as the target platform and the packages that should be used.

The **ecosconfig** tool is also used subsequently to generate a build tree for a configuration. Once a build tree exists, it is possible to run **ecosconfig** again inside the same build tree. This will be necessary if your wish to change some of the configuration options.

ecosconfig does not generate the top-level directory of the build tree; you must do this yourself.

```
$ mkdir ecos-work
$ cd ecos-work
```

The next step is to run **ecosconfig**:

```
$ ecosconfig <qualifiers> <command>
```

ecosconfig qualifiers

The available command line qualifiers for **ecosconfig** are as follows. Multiple qualifiers may be used on the command line:

--help   Provides basic usage guidelines for the available commands and qualifiers.

--config=<file>   Specifies an eCos configuration save file for use by the tool. By default, the file `ecos.ecc` in the current directory is used. Developers may prefer to use a common location for all their eCos configurations rather than keep the configuration information in the base of the build tree.

--prefix=<dir>   Specifies an alternative location for the install tree. By default, the install tree resides inside the `install` directory in the build tree. Developers may prefer to locate the build tree in a temporary file hierarchy but keep the install tree in a more permanent location.

--srcdir=<dir>   Specifies the location of the component repository. By default, the tool uses the location specified in the *ECOS_REPOSITORY* environment variable. Developers may prefer to use of this qualifier if they are working with more than one repository.

--builddir=<dir>   Specifies an alternative location for the build directory. By default, the build directory will be created inside the current directory. Developers may prefer to locate the build tree in a temporary file hierarchy but keep the install tree in a more permanent location.

--no-resolve   Disables the implicit resolution of conflicts while manipulating the configuration data. developers may prefer to resolve conflicts by editing the eCos configuration save file manually.

--ignore-errors   By default, ecosconfig will exit with an error code if the current configuration contains any
-i   conflicts, and it is not possible to generate or update a build tree for such configurations. This qualifier causes ecosconfig to ignore such problems, and hence it is possible to generate a build tree even if there are still conflicts. Of course, there are no guarantees that the resulting system will actually do anything.

--compat   This option invokes the compatability mode of **ecosconfig** with the GUI **configtool**. It sets
-c   both the build and install directories of **ecosconfig** to the directories, relative to the configuration file, that would be created and used by the GUI **configtool** respectively. This option therefore cannot be used in conjuction with either the `--prefix=<dir>` or `--build-dir=<dir>` options. This option is useful to application developers that regularly switch between using the CLI and GUI configuration tools.

--long   Display long listing of packages, templates, targets and attributes.
-l

--verbose   Display more information.
-v

--quiet   Display less information.
-q

The `--config`, `--prefix` and `--srcdir` qualifiers can also be written with two arguments, for example:

```
ecosconfig --srcdir <dir> ...
```

This simplifies filename completion with some shells.

ecosconfig commands

The available commands for **ecosconfig** are as follows:

list [<packages> | <templates> | Lists the available packages, targets and templates as installed in the eCos repository. The ad-
<targets>]   ditional options `<packages>`, `<templates>` and `<targets>` restrict output to pack-

ages, templates and targets respectively and include the `--long` option support to provide additional information. Aliases and package versions are also reported.

new <target> [<template> [<version>]]

Creates a new eCos configuration for the specified target hardware and saves it. A software template may also be specified. By default, the template named 'default, is used. If the template version is not specified, the latest version is used.

target <target>

Changes the target hardware selection for the eCos configuration. This has the effect of unloading packages supporting the target selected previously and loading the packages which support the new hardware. This command will be used typically when switching between a simulator and real hardware.

> ### Note
>
> When switching targets between minor CPU variants, users are advised that inferred changes within common hardware packages are not reset or re-inferred unless a conflict arises as a result of the switch. For example, while a template may contain the SPI package, this package may be disabled in a configuration through inference as a result of the initial hardware lacking an SPI device. However, when switching to a processor variant with more cells (device support), such as one that includes an SPI device, the SPI package could remain disabled when the configuration is switched to the more superior variant unless it is marked as required within the CDL (i.e. it would conflict). Users are therefore advised that switching targets within a configuration may have unanticipated results and are recommended to use the process described in the section called "Switching Targets, Repositories and Versions".

template <template> [<version>]

Changes the template selection for the eCos configuration. This has the effect of unloading packages specified by the template selected previously and loading the packages specified by the new template. By default, the latest version of the specified template is used.

add <packages>

Adds the specified packages to the eCos configuration. This command will be used typically when the template on which a configuration is based does not contain all the packages which are required.For example, add-on packages provided by third parties will not be known to the standard templates, so they will have to be added explicitly.

remove <packages>

Removes the specified packages from the eCos configuration. This command will be used typically when the template on which a configuration is based contains packages which are not required.

packages [loaded | active]

Provides package information regarding the eCos configuration rather than the eCos repository (as provided by the **list** command). The optional argument `loaded` (the default) will list what packages have been loaded into the eCos configuration while `active` will list active packages. The `-- long` option may also be included to provide additional information.

present <package> [<package> ...]

Tests for the presense of the packages and returns a status code of 0 if *all* the packages are present in the eCos configuration, or an error if not. This is useful for automated scripting of the creation of eCos configurations from the command line.

version <version> <packages>

Selects the specified version of a number of packages in the eCos configuration. By default, the most recent version of each package is used. This command will be used typically when an older version of a package is required.

check

Presents the following information concerning the current configuration:

1. the selected target hardware

2. the selected template

3. additional packages

4. removed packages

5. the selected version of packages where this is not the most recent version

6. conflicts in the current configuration

resolve Resolves conflicts identified in the current eCos configuration by invoking an inference capability. Resolved conflicts are reported, but not all conflicts may be resolvable. This command will be used typically following manual editing of the configuration.

export <file> Exports a minimal eCos configuration save file with the specified name. This file contains only those options which do not have their default value. Such files are used typically to transfer option values from one configuration to another.

import <file> Imports a minimal eCos configuration save file with the specified name. The values of those options specified in the file are applied to the current configuration.

tree Generates a build tree based on the current eCos configuration. This command will be used typically just before building eCos.Normally a build tree can only be generated if if the configuration has no unresolved conflicts, but `--ignore-errors` can be used to override this.

get_comment User comments within eCos configurations are not preserved when modified with the **configtool** or **ecosconfig**. However, from eCosPro version 4.0 and above, the eCos Configuration file (normal prefix `<.ecc>`) the eCos Configuration file can contain user comments. The are located at the top of the Configuration file and can be fetched with this command. For example, both tools use User comments to store additional matedata information such as the name of the eCosPro profile used to generate the configuration. If no profile information is provided to either the **configtool** or **ecosconfig**, both tools will use this comment to set a default profile for the configuration of the profile is valid.

add_comment <comment> Append the comment `<comment>` to the user comments within the eCos configuration file.

set_comment <comment> Replaces the user comments within the eCos configuration file with `<comment>`. This will erase any profile indicators stored within the configuration file when the eCos configuration was created.

Conflicts and constraints

Configuration options are not completely independent. For example the C library's `strtod()` and `atof()` functions rely on the math library package to provide certain functionality. If the math library package is removed then the C library can no longer provide these functions. Each package describes constraints like these in CDL *"requires"* properties. If a constraint is not satisfied, then the configuration contains a conflict. For any given conflict there can be several resolution options. For example, it would be possible to add the math library package back to the configuration, or to disable the `strtod()` and `atof()` functions.

The eCos configuration tools will report any conflicts in the current configuration. If there are any such conflicts then the configuration is usually unsafe and it makes no sense to build and run eCos in such circumstances. In fact, any attempt at building eCos is likely to fail. In exceptional cases it is possible to override this by using e.g. the `--ignore-errors` qualifier with ecosconfig.

Many constraints are fairly simple in nature, and the configuration tools contain an inference engine which can resolve the associated conflicts automatically. For example, if the math library package is removed then the inference engine can resolve the resulting

conflict by disabling the configuration option for `strtod()` and `atof()`. All such changes will be reported. Sometimes the inference engine cannot resolve a conflict, for example it is not allowed to override a change that has been made explicitly by the user. Sometimes it will find a solution which does not match the application's requirements.

A typical session involving conflicts would look something like this:

```
$ ecosconfig new pid
```

This creates a new configuration with the default template. For most targets this will not result in any conflicts, because the default settings for the various options meet the requirements of the default template.

For some targets there may be conflicts and the inference engine would come into play.

```
$ ecosconfig remove libm
U CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, new inferred value 0
U CYGFUN_LIBC_strtod, new inferred value 0
U CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, new inferred value 0
```

ecosconfig reports that this change caused three conflicts, all in the C library. The inference engine was able to resolve all the conflicts and update the relevant configuration options accordingly.

To suppress the inference engine `--no-resolve` can be used:

```
$ ecosconfig new pid
$ ecosconfig --no-resolve remove libm
C CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, "requires" constraint not satisfied:  CYGPKG_LIBM
C CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, "requires" constraint not satisfied: CYGPKG_LIBM
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
```

Three unresolved conflicts are reported.

The **check** command can be used to get the current state of the configuration, and the `--verbose` qualifier will provide additional information:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc --verbose check
Target: pid
Template: default
Removed:
  CYGPKG_LIBM
3 conflict(s):
  C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
Possible solution:
  CYGFUN_LIBC_strtod -> 0
  CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT -> 0
  C CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, "requires" constraint not satisfied: CYGPKG_LIBM
Possible solution:
  CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT -> 0
  C CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, "requires" constraint not satisfied:  CYGPKG_LIBM
Possible solution:
  CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT -> 0
```

If the proposed solutions are acceptable, the resolve command can be used to apply them:

```
$ ecosconfig resolve
U CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, new inferred value 0
U CYGFUN_LIBC_strtod, new inferred value 0
U CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, new inferred value 0
```

The current configuration is again conflict-free and it is possible to generate a build tree. The `--quiet` qualifier can be used to suppress the change messages, if desired.

When changing individual configuration options by editing the ecos.ecc file (as described below), the resulting system should be checked and any problems should be resolved. For example, if CYGFUN_LIBC_strtod is explicitly enabled in the savefile:

```
$ edit ecos.ecc
$ ecosconfig check
Target: pid
Template: default
Removed:
CYGPKG_LIBM
1 conflict(s):
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
$ ecosconfig resolve
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
```

In this case the inference engine cannot resolve the conflict automatically because that would involve changing a user setting. Any attempt to generate a build tree will fail:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc tree
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
Unable to generate build tree, this configuration still contains conflicts.
Either resolve the conflicts or use --ignore-errors
```

It is still possible to generate a build tree:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc --ignore-errors tree
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
$ make
```

In this case eCos will fail to build. In other cases of unresolved conflicts eCos may build, but may not run. In general all conflicts should be resolved by editing the `ecos.ecc` file, by letting the inference engine make appropriate changes, or by other means, before any attempt is made to build or run eCos.

Building the System

Once a build tree has been generated with **ecosconfig**, building eCos is straightforward:

```
$ make
```

The build tree contains the subdirectories, makefiles, and everything else that is needed to generate the default configuration for the selected architecture and platform. The only requirement is that the tools needed for that architecture, for example **powerpc-eabi-g++**, are available using the standard search path. If this is not the case then the **make** will fail with an error message. If you have a multiprocessor system then it may be more efficient to use:

```
$ make -j n
```

where n is equal to the number of processors on your system.

Once the **make** process has completed, the install tree will contain the header files and the target library that are needed for application development.

It is also possible to build the system,s test cases for the current configuration:

```
$ make tests
```

The resulting test executables will end up in a `tests` subdirectory of the install tree.

If disk space is scarce then it is possible to make the copy of the install tree for application development purposes, and then use:

```
$ make clean
```

The build tree will now use up a minimum of disk space — the bulk of what is left consists of configuration header files that you may have edited and hence should not be deleted automatically. However, it is possible to rebuild the system at any time without re-invoking **ecosconfig**, just by running **make** again.

Under exceptional circumstances it may be necessary to run **make clean** for other reasons, such as when a new release of the toolchain is installed. The toolchain includes a number of header files which are closely tied to the compiler, for example `limits.h`, and these header files are not and should not be duplicated by eCos. The makefiles perform header file dependency analysis, so that when a header file is changed all affected sources will be rebuilt during the next **make**. This is very useful when the configuration header files are changed, but it also means that a build tree containing information about the locations of header files must be rebuilt. If a new version of the toolchain is installed and the old version is removed then this location information is no longer accurate, and **make** will complain that certain dependencies cannot be satisfied. Under such circumstances it is necessary to do a **make clean** first.

Packages

eCos is a component architecture. The system comes as a number of packages which can be enabled or disabled as required, and new packages can be added as they become available. Unfortunately, the packages are not completely independent: for example the µITRON compatibility package relies almost entirely on functionality provided by the kernel package, and it would not make sense to try to build µITRON if the kernel was disabled. The C library has fewer dependencies: some parts of the C library rely on kernel functionality, but it is possible to disable these parts and thus build a system that has the C library but no kernel. The **ecosconfig** tool has the capability of checking that all the dependencies are satisfied, but it may still be possible to produce configurations that will not build or (conceivably) that will build but not run. Developers should be aware of this and take appropriate care.

By default, **ecosconfig** will include all packages that are appropriate for the specified hardware in the configuration. The common HAL package and the eCos infrastructure must be present in every configuration. In addition, it is always necessary to have one architectural HAL package and one platform HAL package. Other packages are optional, and can be added or removed from a configuration as required.

The application may not require all of the packages; for example, it might not need the µITRON compatibility package, or the floating point support provided by the math library. There is a slight overhead when eCos is built because the packages will get compiled, and there is also a small disk space penalty. However, any unused facilities will get stripped out at link- time, so having redundant packages will not affect the final executable.

Coarse-grained Configuration

Coarse-grained configuration of an eCos system means making configuration changes using the **ecosconfig** tool. These changes include:

1. switching to different target hardware

2. switching to a different template

3. adding or removing a package

4. changing the version of a package

Whenever **ecosconfig** generates or updates an eCos configuration, it generates a configuration save file.

Suppose that the configuration was first created using the following command line:

```
$ ecosconfig new stdeval1
```

To change the target hardware to the Cogent CMA28x PowerPC board, the following command would be needed:

```
$ ecosconfig target cma28x
```

To switch to the PowerPC simulator instead:

```
$ ecosconfig target psim
```

As the hardware changes, hardware-related packages such as the HAL packages and device drivers will be added to and removed from the configuration as appropriate.

> **Note**
>
> See the note on unintended consequences in the section called "Switching Targets, Repositories and Versions".

To remove any package from the current configuration, use the **remove** command:

```
$ ecosconfig remove uitron
```

You can disable multiple packages using multiple arguments, for example:

```
$ ecosconfig remove uitron libm
```

If this turns out to have been a mistake then you can re-enable one or more packages with the **add** command:

```
$ ecosconfig add libm
```

Changing the desired version for a package is also straightforward:

```
$ ecosconfig version v4_0_1 kernel
```

It is necessary to regenerate the build tree and header files following any changes to the configuration before rebuilding eCos:

```
$ ecosconfig tree
```

Fine-grained Configuration

ecosconfig only provides coarse-grained control over the configuration: the hardware, the template and the packages that should be built. Unlike the Configuration Tool, **ecosconfig** does not provide any facilities for manipulating finer-grained configuration options such as how many priority levels the scheduler should support. There are hundreds of these options, and manipulating them by means of command line arguments would not be sensible.

In the current system fine-grained configuration options may be manipulated by manual editing of the configuration file. When a file has been edited in this way, the **ecosconfig** tool should be used to check the configuration for any conflicts which may have been introduced:

```
$ ecosconfig check
```

The **check** command will list all conflicts and will also rewrite the configuration file, propagating any changes which affect other options. The user may choose to resolve the conflicts either by re-editing the configuration file manually or by invoking the inference engine using the **resolve** command:

```
$ ecosconfig resolve
```

The **resolve** command will list all conflicts which can be resolved and save the resulting changes to the configuration.

It is necessary to regenerate the build tree and header files following any changes to the configuration before rebuilding eCos:

```
$ ecosconfig tree
```

All the configuration options and their descriptions are listed in the *eCos Reference Manual*.

Editing an eCos Savefile

The eCos configuration information is held in a single savefile, typically `ecos.ecc`, which can be generated by either the GUI configuration tool or by the command line **ecosconfig** tool. The file normally exists at the top level of the build tree. It is a text file, allowing the various configurations options to be edited inside a suitable text editor or by other programs or scripts, as well as in the GUI config tool.

An eCos savefile is actually a script in the *Tcl* programming language, so any modifications to the file need to preserve Tcl syntax. For most configuration options, any modifications will be trivial and there is no need to worry about Tcl syntax. For example,

changing a 1 to a 0 to disable an option. For more complicated options, for example `CYGDAT_UITRON_TASK_EXTERNS`, which involves some lines of C code, more care has to be taken. If an edited savefile is no longer a valid Tcl script then the configuration tools will be unable to read back the data for further processing, for example to generate a build tree. An outline of Tcl syntax is given below. One point worth noting here is that a line that begins with a "#" is usually a comment, and the bulk of an eCos savefile actually consists of such comments, to make it easier to edit.

Header

An eCos savefile begins with a header, which typically looks something like this:

```
# eCos saved configuration
# ---- commands ---------------------------------------------------
# This section contains information about the savefile format.
# It should not be edited. Any modifications made to this section
# may make it impossible for the configuration tools to read
# the savefile.
cdl_savefile_version 1;
cdl_savefile_command cdl_savefile_version {};
cdl_savefile_command cdl_savefile_command {};
cdl_savefile_command
cdl_configuration { description hardware template package };
cdl_savefile_command cdl_package { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_component { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_option { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_interface { value_source user_value wizard_value inferred_value };
```

This section of the savefile is intended for use by the configuration system, and should not be edited. If this section is edited then the various configuration tools may no longer be able to read in the modified savefile.

Toplevel Section

The header is followed by a section that defines the configuration as a whole. A typical example would be:

```
# ---- toplevel ---------------------------------------------------
# This section defines the toplevel configuration object. The only
# values that can be changed are the name of the configuration and
# the description field. It is not possible to modify the target,
# the template or the set of packages simply by editing the lines
# below because these changes have wide-ranging effects. Instead
# the appropriate tools should be used to make such modifications.
cdl_configuration eCos {
  description "" ;
  # These fields should not be modified.
  hardware    pid ;
  template    uitron ;
  package -hardware CYGPKG_HAL_ARM current ;
  package -hardware CYGPKG_HAL_ARM_PID current ;
  package -hardware CYGPKG_IO_SERIAL current ;
  package -template CYGPKG_HAL current ;
  package -template CYGPKG_IO current ;
  package -template CYGPKG_INFRA current ;
  package -template CYGPKG_KERNEL current ;
  package -template CYGPKG_UITRON current ;
  package -template CYGPKG_LIBC current ;
  package -template CYGPKG_LIBM current ;
  package -template CYGPKG_DEVICES_WALLCLOCK current ;
  package -template CYGPKG_ERROR current ;
};
```

This section allows the configuration tools to reload the various packages that make up the configuration. Most of the information should not be edited. If it is necessary to add a new package or to remove an existing one then the appropriate tools should be used for this, for example:

```
$ ecosconfig remove CYGPKG_LIBM
```

There are two fields which can be edited. Configurations have a name; in this case eCos. They can also have a description, which is some arbitrary text. The configuration tools do not make use of these fields, they exist so that users can store additional information about a configuration.

Conflicts Section

The toplevel section is followed by details of all the conflicts (if any) in the configuration, for example:

```
# ---- conflicts ----------------------------------------------------
# There are 2 conflicts.
#
# option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET
#   Property LegalValues
#   Illegal current value 100000
#   Legal values are: -90000 to 90000
#
# option CYGSEM_LIBC_TIME_CLOCK_WORKING
#   Property Requires
#   Requires constraint not satisfied: CYGFUN_KERNEL_THREADS_TIMER
```

When editing a configuration you may end up with something that is invalid. Any problems in the configuration will be reported in the conflicts section. In this case there are two conflicts. The option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET has been given an illegal value: typically this would be fixed by searching for the definition of that option later on in the savefile and modifying the value. The second conflict is more interesting, an unsatisfied *requires* constraint. Configuration options are not independent: disabling some functionality in, say, the kernel, can have an impact elsewhere; in this case the C library. The various dependencies between the options are specified by the component developers and checked by the configuration system. In this case there are two obvious ways in which the conflict could be resolved: re- enabling CYGFUN_KERNEL_THREADS_TIMER, or disabling CYGSEM_LIBC_TIME_CLOCK_WORKING. Both of these options will be listed later on in the file.

Some care has to be taken when modifying configuration options, to avoid introducing new conflict. For instance it is possible that there might be other options in the system which have a dependency on CYGSEM_LIBC_TIME_CLOCK_WORKING, so disabling that option may not be the best way to resolve the conflict. Details of all such dependencies are provided in the appropriate places in the savefile.

It is not absolutely required that a configuration be conflict- free before generating a build tree and building eCos. It is up to the developers of each component to decide what would happen if an attempt is made to build eCos while there are still conflicts. In serious cases there is likely to be a compile-time failure, or possibly a link-time failure. In less serious cases the system may build happily and the application can be linked with the resulting library, but the component may not quite function as intended - although it may still be good enough for the specific needs of the application. It is also possible that everything builds and links, but once in a while the system will unaccountably crash. Using a configuration that still has conflicts is done entirely at the user's risk.

Data Section

The bulk of the savefile lists the various packages, components, and options, including their values and the various dependencies. A number of global options come first, especially those related to the build process such as compiler flags. These are followed by the various packages, and the components and options within those packages, in order.

Packages, components and options are organized in a hierarchy. If a particular component is disabled then all options and sub-components below it will be inactive: any changes made to these will have no effect. The savefile contains information about the hierarchy in the form of comments, for example:

```
cdl_package CYGPKG_KERNEL ...
# >
cdl_component CYGPKG_KERNEL_EXCEPTIONS ...
# >
cdl_option CYGSEM_KERNEL_EXCEPTIONS_DECODE ...
```

```
cdl_option CYGSEM_KERNEL_EXCEPTIONS_GLOBAL ...
# <
cdl_component CYGPKG_KERNEL_SCHED ...
# >
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE ...
cdl_option CYGSEM_KERNEL_SCHED_BITMAP ...
# <
# <
```

This corresponds to the following hierarchy:

```
CYGPKG_KERNEL
  CYGPKG_KERNEL_EXCEPTIONS
    CYGSEM_KERNEL_EXCEPTIONS_DECODE
    CYGSEM_KERNEL_EXCEPTIONS_GLOBAL
  CYGPKG_KERNEL_SCHED
    CYGSEM_KERNEL_SCHED_MLQUEUE
    CYGSEM_KERNEL_SCHED_BITMAP
```

Providing the hierarchy information in this way allows programs or scripts to analyze the savefile and readily determine the hierarchy. It could also be used by a sufficiently powerful editor to support structured editing of eCos savefiles. The information is not used by the configuration tools themselves since they obtain the hierarchy from the original CDL scripts.

Each configurable entity is preceded by a comment, of the following form:

```
# Kernel schedulers
# doc: ref/ecos-ref/ecos-kernel-overview.html#THE-SCHEDULER
# The eCos kernel provides a choice of schedulers. In addition
# there are a number of configuration options to control the
# detailed behaviour of these schedulers.
cdl_component CYGPKG_KERNEL_SCHED {
  ...
};
```

This provides a short textual alias `Kernel schedulers` for the component. If online documentation is available for the configurable entity then this will come next. Finally there is a short description of the entity as a whole. All this information is provided by the component developers.

Each configurable entity takes the form:

```
<type> <name> {
  <data>
};
```

Configurable entities may not be active. This can be either because the parent is disabled or inactive, or because there are one or more *active_if* properties. Modifying the value of an inactive entity has no effect on the configuration, so this information is provided first:

```
cdl_option CYGSEM_KERNEL_EXCEPTIONS_DECODE {
  # This option is not active
  # The parent CYGPKG_KERNEL_EXCEPTIONS is disabled
  ...
};
...
cdl_option CYGIMP_IDLE_THREAD_YIELD {
  # This option is not active
  # ActiveIf constraint: (CYGNUM_KERNEL_SCHED_PRIORITIES == 1)
  #     CYGNUM_KERNEL_SCHED_PRIORITIES == 32
  #   --> 0
  ...
};
```

For `CYGIMP_IDLE_THREAD_YIELD` the savefile lists the expression that must be satisfied if the option is to be active, followed by the current value of all entities that are referenced in the expression, and finally the result of evaluating that expression.

Not all options are directly modifiable in the savefile. First, the value of packages (which is the version of that package loaded into the configuration) cannot be modified here.

```
cdl_package CYGPKG_KERNEL {
  # Packages cannot be added or removed, nor can their version be changed,
  # simply by editing their value. Instead the appropriate configuration
  # should be used to perform these actions.
  ...
};
```

The version of a package can be changed using e.g.:

```
$ ecosconfig version 1.3 CYGPKG_KERNEL
```

Even though a package‚s value cannot be modified here, it is still important for the savefile to contain the details. In particular packages may impose constraints on other configurable entities and may be referenced by other configurable entities. Also it would be difficult to understand or extract the configuration‚s hierarchy if the packages were not listed in the appropriate places in the savefile.

Some components (or, conceivably, options) do not have any associated data. Typically they serve only to introduce another level in the hierarchy, which can be useful in the context of the GUI configuration tool.

```
cdl_component CYGPKG_HAL_COMMON_INTERRUPTS {
  # There is no associated value.
};
```

Other components or options have a calculated value. These are not user-modifiable, but typically the value will depend on other options which can be modified. Such calculated options can be useful when controlling what gets built or what ends up in the generated configuration header files. A calculated value may also effect other parts of the configuration, for instance, via a *requires* constraint.

```
cdl_option BUFSIZ {
  # Calculated value: CYGSEM_LIBC_STDIO_WANT_BUFFERED_IO ? CYGNUM_LIBC_STDIO_BUFSIZE : 0
  #     CYGSEM_LIBC_STDIO_WANT_BUFFERED_IO == 1
  #     CYGNUM_LIBC_STDIO_BUFSIZE == 256
  # Current_value: 256
};
```

A special type of calculated value is the *interface*. The value of an interface is the number of active and enabled options which *implement* that interface. Again the value of an interface cannot be modified directly; only by modifying the options which implement the interface. However, an interface can be referenced by other parts of the configuration.

```
cdl_interface CYGINT_KERNEL_SCHEDULER {
  # Implemented by CYGSEM_KERNEL_SCHED_MLQUEUE, active, enabled
  # Implemented by CYGSEM_KERNEL_SCHED_BITMAP, active, disabled
  # This value cannot be modified here.
  # Current_value: 1
  # Requires: 1 == CYGINT_KERNEL_SCHEDULER
  #     CYGINT_KERNEL_SCHEDULER == 1
  #   --> 1
  # The following properties are affected by this value
  # interface CYGINT_KERNEL_SCHEDULER
  #     Requires: 1 == CYGINT_KERNEL_SCHEDULER
};
```

If the configurable entity is modifiable then there will be lines like the following:

```
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE {
  ...
  # Flavor: bool
  # No user value, uncomment the following line to provide one.
  # user_value 1
  # value_source default
```

```
  # Default value: 1
  ...
};
```

Configurable entities can have one of four different flavors: none, bool, data and booldata. Flavor none indicates that there is no data associated with the entity, typically it just acts as a placeholder in the overall hierarchy. Flavor bool is the most common, it is a simple yes-or-no choice. Flavor data is for more complicated configuration choices, for instance the size of an array or the name of a device. Flavor booldata is a combination of bool and data: the option can be enabled or disabled, and there is some additional data associated with the option as well.

In the above example the user has not modified this particular option, as indicated by the comment and by the commented-out user_value line. To disable this option the file should be edited to:

```
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE {
  ...
  # Flavor: bool
  # No user value, uncomment the following line to provide one.
  user_value 0
  # value_source default
  # Default value: 1
  ...
}
```

The comment preceding the user_value 0 line can be removed if desired, otherwise it will be removed automatically the next time the file is read and updated by the configuration tools.

Much the same process should be used for options with the data flavor, for example:

```
cdl_option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET {
  # Flavor: data
  # No user value, uncomment the following line to provide one.
  # user_value 3600
  # value_source default
  # Default value: 3600
  # Legal values: -90000 to 90000
};
```

can be changed to:

```
cdl_option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET {
  # Flavor: data
  user_value 7200
  # value_source default
  # Default value: 3600
  # Legal values: -90000 to 90000
};
```

Note that the original text provides the default value in the user_value comment, on the assumption that the desired new value is likely to be similar to the default value. The value_source comment does not need to be updated, it will be fixed up if the savefile is fed back into the configuration system and regenerated.

For options with the booldata flavor, the user_value line needs take two arguments. The first argument is for the boolean part, the second for the data part. For example:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
  # Flavor: booldata
  # No user value, uncomment the following line to provide one.
  # user_value 1 POSIX
  # value_source default
  # Default value: 1 POSIX
  # Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
  ...
};
```

could be changed to:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
  # Flavor: booldata
  user_value 1 IEEE
  # value_source default
  # Default value: 1 POSIX
  # Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
  ...
};
```

or alternatively, if the whole component should be disabled, to:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
  # Flavor: booldata
  user_value 0 POSIX
  # value_source default
  # Default value: 1 POSIX
  # Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
  ...
};
```

Some options take values that span multiple lines. An example would be:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_INITIALIZERS {
  # Flavor: data
  # No user value, uncomment the following line to provide one.
  # user_value \
  # "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \\
  #  CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \\
  #  CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
  # value_source default
  # Default value: \
  #     "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \\
  #       CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \\
  #       CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
};
```

Setting a user value for this option involves uncommenting and modifying all relevant lines, for example:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_INITIALIZERS {
  # Flavor: data
  user_value \
  "CYG_UIT_MEMPOOLVAR( vpool1, 4000 ), \\
  CYG_UIT_MEMPOOLVAR( vpool2, 4000 ),"
  # value_source default
  # Default value: \
  #     "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \\
  #       CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \\
  #       CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
};
```

In such cases appropriate care has to be taken to preserve Tcl syntax, as discussed below.

The configuration system has the ability to keep track of several different values for any given option. All options start off with a default value, in other words their value source is set to `default`. If a configuration involves conflicts and the configuration system,s inference engine is allowed to resolve these automatically then it may provide an `inferred` value instead, for example:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT {
  # Flavor: bool
  # No user value, uncomment the following line to provide one.
  # user_value 1
  # The inferred value should not be edited directly.
  inferred_value 0
  # value_source inferred
  # Default value: 1
```

```
  ...
};
```

Inferred values are calculated by the configuration system and should not be edited by the user. If the inferred value is not correct then a user value should be substituted instead:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT {
  # Flavor: bool
  user_value 1
  # The inferred value should not be edited directly.
  inferred_value 0
  # value_source inferred
  # Default value: 1
  ...
};
```

The inference engine will not override a user value with an inferred one. Making a change like this may well re-introduce a conflict, since the inferred value was only calculated to resolve a conflict. Another run of the inference engine may find a different and more acceptable way of resolving the conflict, but this is not guaranteed and it may be up to the user to examine the various dependencies and work out an acceptable solution.

Inferred values are listed in the savefile because the exact inferred value may depend on the order in which changes were made and conflicts were resolved. If the inferred values were absent then it is possible that reloading a savefile would not exactly restore the configuration. Default values on the other hand are entirely deterministic so there is no actual need for the values to be listed in the savefile. However, the default value can be very useful information so it is provided in a comment.

Occasionally the user will want to do some experimentation, and temporarily switch an option from a user value back to a default or inferred one to see what the effect would be. This could be achieved by simply commenting out the user value. For instance, if the current savefile contains:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT {
  # Flavor: bool
  user_value 1
  # The inferred value should not be edited directly.
  inferred_value 0
  # value_source user
  # Default value: 1
  ...
};
```

then the inferred value could be restored by commenting out or removing the `user_value` line:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT {
  # Flavor: bool
  # user_value 1
  # The inferred value should not be edited directly.
  inferred_value 0
  # value_source user
  # Default value: 1
  ...
};
```

This is fine for simple values. However if the value is complicated then there is a problem: commenting out the `user_value` line or lines means that the user value becomes invisible to the configuration system, so if the savefile is loaded and then regenerated the information will be lost. An alternative approach is to keep the `user_value` but explicitly set the `value_source` line, for example:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT {
  # Flavor: bool
  user_value 1
  # The inferred value should not be edited directly.
  inferred_value 0
  value_source inferred
  # Default value: 1
```

```
  ...
};
```

In this case the configuration system will use the inferred value for the purposes of dependency analysis etc., even though a user value is present. To restore the user value the `value_source` line can be commented out again. If there is no explicit `value_source` then the configuration system will just use the highest priority one: the user value if it exists; otherwise the inferred value if it exists; otherwise the default value which always exists.

The default value for an option can be a simple constant, or it can be an expression involving other options. In the latter case the expression will be listed, together with the values for all options referenced in the expression and the current result of evaluating that expression. This is for informational purposes only, the default value is always recalculated deterministically when loading in a savefile.

```
cdl_option CYGBLD_GLOBAL_COMMAND_PREFIX {
  # Flavor: data
  # No user value, uncomment the following line to provide one.
  # user_value arm-elf
  # value_source default
  # Default value:  CYGHWR_THUMB ? "thumb-elf" : "arm-elf"
  #     CYGHWR_THUMB == 0
  #   --> arm-elf
};
```

For options with the data or booldata flavor, there are likely to be constraints on the possible values. If the value is supposed to be a number in a given range and a user value of "`hello world`" is provided instead then there are likely to be compile-time failures. Component developers can specify constraints on the legal values, and these will be listed in the savefile.

```
cdl_option X_TLOSS {
  # Flavor: data
  # No user value, uncomment the following line to provide one.
  # user_value 1.41484755040569E+16
  # value_source default
  # Default value: 1.41484755040569E+16
  # Legal values: 1 to 1e308
};
```

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
  # Flavor: booldata
  # No user value, uncomment the following line to provide one.
  # user_value 1 POSIX
  # value_source default
  # Default value: 1 POSIX
  # Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
  ...
};
```

In some cases the legal values list may be an expression involving other options. If so then the current values of the referenced options will be listed, together with the result of evaluating the list expression, just as for default value expressions.

If an illegal value is provided then this will result in a conflict, listed in the conflicts section of the savefile. For more complicated options a simple legal values list is not sufficient to allow the current value to be validated, and the configuration system will be unable to flag conflicts. This issue will be addressed in future releases of the configuration system.

Following the value-related fields for a given option, any *requires* constraints belonging to this option will be listed. These constraints are only effective if the option is active and, for bool and booldata flavors, enabled. If some aspect of eCos functionality is inactive or disabled then it cannot impose any constraints on the rest of the system. As usual, the full expression will be listed followed by the current values of all options that are referenced and the result of evaluating the expression:

```
cdl_option CYGSEM_LIBC_TIME_TIME_WORKING {
  ...
  # Requires: CYGPKG_DEVICES_WALLCLOCK
  #     CYGPKG_DEVICES_WALLCLOCK == current
  #   --> 1
```

```
};
```

When modifying the value of an option it is useful to know not only what constraints the option imposes on the rest of the system but also what other options in the system depend in some way on this one. The savefile provides this information:

```
cdl_option CYGFUN_KERNEL_THREADS_TIMER {
  ...
  # The following properties are affected by this value
  # option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT
  #     Requires: CYGFUN_KERNEL_THREADS_TIMER
  # option CYGIMP_UITRON_STRICT_CONFORMANCE
  #     Requires: CYGFUN_KERNEL_THREADS_TIMER
  # option CYGSEM_LIBC_TIME_CLOCK_WORKING
  #     Requires: CYGFUN_KERNEL_THREADS_TIMER
};
```

Tcl Syntax

eCos savefiles are implemented as Tcl scripts, and are read in by running the data through a standard Tcl interpreter that has been extended with a small number of additional commands such as `cdl_option` and `cdl_configuration`. In many cases this is an implementation detail that can be safely ignored while editing a savefile: simply replacing a `1` with a `0` to disable some functionality is not going to affect whether or not the savefile is still a valid Tcl script and can be processed by a Tcl interpreter. However, there are more complicated cases where an understanding of Tcl syntax is at least desirable, for example:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_EXTERNS {
  # Flavor: data
  user_value \
  "static char vpool1\[ 2000 \], \\
  vpool2\[ 2000 \], \\
  vpool3\[ 2000 \];"
  # value_source default
  # Default value: \
  #     "static char vpool1\[ 2000 \], \\
  #       vpool2\[ 2000 \], \\
  #       vpool3\[ 2000 \];"
};
```

The backslash at the end of the `user_value` line is processed by the Tcl interpreter as a line continuation character. The quote marks around the user data are also interpreted by the Tcl interpreter and serve to turn the entire data field into a single argument. The backslashes preceding the opening and closing square brackets prevent the Tcl interpreter from treating these characters specially, otherwise there would be an attempt at *command substitution* as described below. The double backslashes at the end of each line of the data will be turned into a single backslash by the Tcl interpreter, rather than escaping the newline character, so that the actual data seen by the configuration system is:

```
static char vpool1[ 2000 ], \
vpool2[ 2000 ], \
vpool3[ 2000 ];
```

This is of course the data that should end up in the µITRON configuration header file. The opening and closing braces surrounding the entire body of the option data are also significant and cause this body to be passed as a single argument to the **cdl_option** command. The closing semicolon is optional in this example, but provides a small amount of additional robustness if the savefile is edited such that it is no longer a valid Tcl script. If the data contained any `$` characters then these would have to be treated specially as well, via a backslash escape.

In spite of what all the above might seem to suggest, Tcl is actually a very simple yet powerful scripting language: the syntax is defined by just eleven rules. On occasion this simplicity means that Tcl,s behaviour is subtly different from other languages, which can confuse newcomers.

When the Tcl interpreter is passed some data such as `puts  Hello`, it splits this data into a command and its arguments. The command will be terminated by a newline or by a semicolon, unless one of the quoting mechanisms is used. The command and each of its arguments are separated by white space. So in the following example:

```
puts Hello
set x 42
```

will result in two separate commands being executed. The first command is `puts` and is passed a single argument, `Hello`. The second command is `set` and is passed two arguments, `x` and `42`. The intervening newline character serves to terminate the first command, and a semi-colon separator could be used instead:

```
puts Hello;set x 42
```

Any white space surrounding the semicolon is just ignored because it does not serve to separate arguments.

Now consider the following:

```
set x Hello world
```

This is not valid Tcl. It is an attempt to invoke the `set` command with three arguments: `x`, `Hello`, and `world`. The `set` only takes two arguments, a variable name and a value, so it is necessary to combine the data into a single argument by quoting:

```
set x "Hello world"
```

When the Tcl interpreter encounters the first quote character it treats all subsequent data up to but not including the closing quote as part of the current argument. The quote marks are removed by the interpreter, so the second argument passed to the `set` command is just `Hello world` without the quote characters. This can be significant in the context of eCos savefiles. For instance, consider the following configuration option:

```
cdl_option CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE {
  # Flavor: data
  # No user value, uncomment the following line to provide one.
  # user_value "\"/dev/ttydiag\""
  # value_source default
  # Default value: "\"/dev/ttydiag\""
};
```

The desired value of the configuration option should be a valid C string, complete with quote characters. If the savefile was edited to:

```
cdl_option CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE {
  # Flavor: data
  user_value "/dev/ttydiag"
  # value_source default
  # Default value: "\"/dev/ttydiag\""
};
```

then the Tcl interpreter would remove the quote marks when the savefile is read back in, so the option,s value would not have any quote marks and would not be a valid C string. The configuration system is not yet able to perform the required validation so the following #define would be generated in the configuration header file:

```
#define CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE /dev/ttydiag
```

This is likely to cause a compile-time failure when building eCos.

A quoted argument continues until the closing quote character is encountered, which means that it can span multiple lines. This can also be encountered in eCos savefiles, for instance, in the CYGDAT_UITRON_MEMPOOLVAR_EXTERNS example mentioned earlier. Newline or semicolon characters do not terminate the current command in such cases.

The Tcl interpreter supports much the same forms of backslash substitution as other common programming languages. Some backslash sequences such as \n will be replaced by the appropriate character. The sequence \\ will be replaced by a single backslash. A backslash at the very end of a line will cause that backslash, the newline character, and any white space at the start of the next line to be replaced by a single space. Hence the following two Tcl commands are equivalent:

```
puts  "Hello\nworld\n"
puts \
  "Hello
  world
```

```
"
```

In addition to quote and backslash characters, the Tcl interpreter treats square brackets, the $ character, and braces specially. Square brackets are used for command substitution, for example:

```
puts "The answer is [expr 6 * 9]"
```

When the Tcl interpreter encounters the square brackets it will treat the contents as another command that should be executed first, and the result of executing that is used when continuing to process the script. In this case the Tcl interpreter will execute the command `expr 6 * 9`, yielding a result of 54, and then the Tcl interpreter will execute `puts "The answer is 54"`. It should be noted that the interpreter contains only one level of substitution: if the result of performing command substitution performs further special characters such as square brackets then these will not be treated specially.

Command line substitution is very unlikely to prove useful in the context of an eCos savefile, but it is part of the Tcl language and hence cannot be easily suppressed while reading in a savefile. As a result care has to be taken when savefile data involves square brackets. Consider the following:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
  ...
  user_value \
  "static char fpool1[ 2000 ],
  fpool2[ 2000 ];"
  ...
};
```

The Tcl interpreter will interpret the square brackets as an attempt at command substitution and hence it will attempt to execute the command `2000` with no arguments. No such command is defined by the Tcl language or by the savefile-related extensions provided by the configuration system, so this will result in an error when an attempt is made to read back the savefile. Instead it is necessary to backslash-escape the square brackets and thus suppress command substitution:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
  ...
  user_value \
  "static char fpool1\[ 2000 \],
  fpool2\[ 2000 \];"
  ...
};
```

Similarly the $ character is used in Tcl scripts to perform variable substitution:

```
set x [expr 6 * 9]
    puts "The answer is $x"
```

Variable substitution, like command substitution, is very unlikely to prove useful in the context of an eCos savefile. Should it be necessary to have a $ character in configuration data then again a backslash escape needs to be used.

```
cdl_option FOODAT_MONITOR_PROMPT {
  ...
  user_value "\$ "
  ...
};
```

Braces are used to collect a sequence of characters into a single argument, just like quotes. The difference is that variable, command and backslash substitution do not occur inside braces (with the sole exception of backslash substitution at the end of a line). So, for example, the CYGDAT_UITRON_MEMPOOL_EXTERNFIXED_EXTERNS value could be written as:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
  ...
  user_value \
  {static char fpool1[ 2000 ],
  fpool2[ 2000 ];}
  ...
};
```

The configuration system does not use this when generating savefiles because for simple edits of a savefile by inexperienced users the use of brace characters is likely to be a little bit more confusing than the use of quotes.

At this stage it is worth noting that the basic format of each configuration option in the savefile makes use of braces:

```
cdl_option <name> {
  ...
};
```

The configuration system extends the Tcl language with a small number of additional commands such as `cdl_option`. These commands take two arguments, a name and a body, where the body consists of all the text between the braces. First a check is made that the specified option is actually present in the configuration. Then the body is executed in a recursive invocation of the Tcl interpreter, this time with additional commands such as `user_value` and `value_source`. If, after editing, the braces are not correctly matched up then the savefile will no longer be a valid Tcl script and errors will be reported when the savefile is loaded again.

Comments in Tcl scripts are introduced by a hash character #. However, a hash character only introduces a comment if it occurs where a command is expected. Consider the following:

```
# This is a comment
puts "Hello" # world
```

The first line is a valid comment, since the hash character occurs right at the start where a command name is expected. The second line does not contain a comment. Instead it is an attempt to invoke the `puts` command with three arguments: `Hello`, `#` and `world`. These are not valid arguments for the `puts` command so an error will be raised.

If the second line was rewritten as:

```
puts "Hello"; # world
```

then this is a valid Tcl script. The semicolon identifies the end of the current command, so the hash character occurs at a point where the next command would start and hence it is interpreted as the start of a comment.

This handling of comments can lead to subtle behaviour. Consider the following:

```
cdl_option WHATEVER {
  # This is a comment }
  user_value 42
  ...
}
```

Consider the way the Tcl interpreter processes this. The command name and the first argument do not pose any special difficulties. The opening brace is interpreted as the start of the next argument, which continues until a closing brace is encountered. In this case the closing brace occurs on the second line, so the second argument passed to `cdl_option` is `\n # This is a comment`. This second argument is processed in a recursive invocation of the Tcl interpreter and does not contain any commands, just a comment. Toplevel savefile processing then resumes, and the next command that is encountered is `user_value`. Since the relevant savefile code is not currently processing a configuration option this is an error. Later on the Tcl interpreter would encounter a closing brace by itself, which is also an error. Fortunately this sequence of events is very unlikely to occur when editing generated savefiles.

This should be sufficient information about Tcl to allow for safe editing of eCos savefiles. Further information is available from a wide variety of sources, for example the book *Tcl and the Tk Toolkit* by John K Ousterhout.

Editing the Sources

For many users, controlling the packages and manipulating the available configuration options will be sufficient to create an embedded operating system that meets the application's requirements. However, since eCos is shipped entirely in source form, it is possible to go further when necessary: you can edit the eCos sources themselves. This requires some understanding of the way the eCos build system works.

The most obvious place to edit the source code is directly in the component repository. For example, you could edit the file `kernel/<version>/src/sync/mutex.cxx` to change the way kernel mutexes work, or possibly just to add some extra diagnostics or assertions. Once the file has been edited, it is possible to invoke **make** at the top level of the build tree and the target library will be rebuilt as required. A small optimization is possible: the build tree is largely a mirror of the component repository, so it too will contain a subdirectory `kernel/<version>`; if make is invoked in this directory then it will only check for changes to the kernel sources, which is a bit more efficient than checking for changes throughout the component repository.

Editing a file in the component repository is fine if this tree is used for only one eCos configuration. If the repository is used for several different configurations, however, and especially if it is shared by multiple users, then making what may be experimental changes to the master sources would be a bad idea. The build system provides an alternative. It is possible to make a copy of the file in the build tree, in other words copy `mutex.cxx` from the `kernel/<version>/src/sync` directory in the component repository to `kernel/<version>/src/sync` in the build tree, and edit the file in the build tree. When **make** is invoked it will pick up local copies of any of the sources in preference to the master versions in the component repository. Once you have finished modifying the eCos sources you can install the final version back in the component repository. If the changes were temporary in nature and only served to aid the debugging process, then you can discard the modified version of the sources.

The situation is slightly more complicated for the header files that a package may export, such as the C library,s `stdio.h` header file, which can be found in the directory `language/c/libc/<version>/include`. If such a header file is changed, either directly in the component repository or after copying it to the build tree, then **make** must be invoked at the top level of the build tree. In cases like this it is not safe to rebuild just the C library because other packages may depend on the contents of `stdio.h`.

Modifying the Memory Layout

Each eCos platform package is supplied with linker script fragments which describe the location of memory regions on the evaluation board and the location of memory sections within these regions. The correct linker script fragment is selected and included in the eCos linker script `target.ld` when eCos is built.

It is not necessary to modify the default memory layouts in order to start development with eCos. However, it will be necessary to edit a linker script fragment when the memory map of the evaluation board is changed. For example, if additional memory is added, the linker must be notified that the new memory is available for use. As a minimum, this would involve modifying the length of the corresponding memory region. Where the available memory is non-contiguous, it may be necessary to declare a new memory region and reassign certain linker output sections to the new region.

Linker script fragments and memory layout header files should be edited within the eCos install tree. They are located at `include/pkgconf/mlt_*.*`. Where multiple start-up types are in use, it will be necessary to edit multiple linker script fragments and header files. The information provided in the header file and the corresponding linker script fragment must always match. A typical linker script fragment is shown below:

Example 26.1. eCos linker script fragment

```
MEMORY
{
  rom : ORIGIN = 0x40000000, LENGTH = 0x80000
  ram : ORIGIN = 0x48000000, LENGTH = 0x200000
}
SECTIONS
{
  SECTIONS_BEGIN
  SECTION_rom_vectors (rom, 0x40000000, LMA_EQ_VMA)
  SECTION_text (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_fini (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_rodata (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_rodata1 (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_fixup (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_gcc_except_table (rom, ALIGN (0x1), LMA_EQ_VMA)
  SECTION_data (ram, 0x48000000, FOLLOWING (.gcc_except_table))
  SECTION_bss (ram, ALIGN (0x4), LMA_EQ_VMA)
  SECTIONS_END
```

```
}
```

The file consists of two blocks, the MEMORY block contains lines describing the address (ORIGIN) and the size (LENGTH) of each memory region. The MEMORY block is followed by the SECTIONS block which contains lines describing each of the linker output sections. Each section is represented by a macro call. The arguments of these macros are ordered as follows:

1. The memory region in which the section will finally reside.

2. The final address (VMA) of the section. This is expressed using one of the following forms:

    n   at the absolute address specified by the unsigned integer *n*

    ALIGN (*n*)   following the final location of the previous section with alignment to the next *n*-byte boundary

3. The initial address (LMA) of the section. This is expressed using one of the following forms:

    LMA_EQ_VMA   the LMA equals the VMA (no relocation)

    AT (*n*)   at the absolute address specified by the unsigned integer *n*

    FOLLOWING (*.name*)   following the initial location of section *name*

In order to maintain compatibility with linker script fragments and header files exported by the eCos Configuration Tool, the use of other expressions within these files is not recommended.

Note that the names of the linker output sections will vary between target architectures. A description of these sections can be found in the specific GCC documentation for your architecture.

Chapter 27. Managing the Package Repository

A source distribution of eCos consists of a number of packages, such as the kernel, the C library, and the µITRON subsystems. These are individually versioned in the tree structure of the source code to support distribution on a per-package basis and to support third party packages whose versioning systems might be different. A command-line tool, **ecosadmin.tcl** is used is used to manage the installation and removal of packages from a variety of sources with potentially multiple versions. The eCos Configuration Tool provides a graphical user interface to this tool through a dialog box as illustrated in Figure 27.1, " Package Administration ".

Notes

1. The presence of the version information in the source tree structure might be a hindrance to the use of source control systems such as *CVS* or *SourceSafe*. To work in this way, you can rename all the version components to a common name such as "current" thus unifying the structure of source trees from distinct eCos releases.

2. The eCos build system will treat any such name as just another version of the package(s), and support building in exactly the same way. However, performing this rename invalidates any existing build trees that referred to the versioned source tree, so do the rename first, before any other work, and do a complete rebuild afterwards. If you do have an existing configuration you wish to move over to a new structure, you are advised to use the process described in the section called "Switching Targets, Repositories and Versions".

3. **ecosadmin.tcl** traditionally accompanies each eCos and eCosPro release at the base of the eCos respository in the `packages` sub-directory, not as part of eCos or eCosPro User or Host Tools releases.

Package Installation

ecosadmin.tcl, the eCos Package Administration Tool, is a command line tool that allows administration of packages within an eCos Repository. This tool is a Tcl script which allows the user to add new eCos packages and new versions of existing packages to an eCos repository. Such packages are distributed as a single file in the eCos package distribution format and, by convention, eCos package distribution files are given the `.epk` suffix. Unwanted packages or versions of packages may also be removed from an eCos repository using this tool, and a GUI interface to this tool provided by the eCos Configuration Tool.

Using the Package Administration Tool

The graphical interface accesses all the functionality of the command-line tool and may be accessed from the menu option Tools → Administration. You will first be prompted to save your existing configuration before the dialog in Figure 27.1, " Package Administration " is displayed.

Figure 27.1. Package Administration
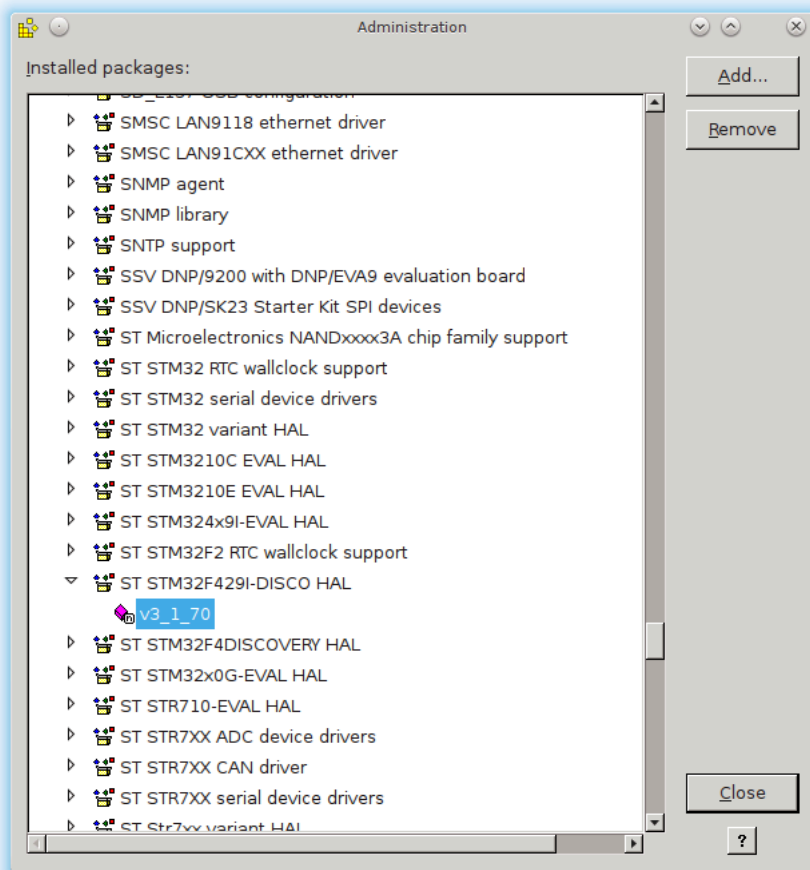
The dialog displays the packages which are currently installed in the form of a tree. The installed versions of each package may be examined by expanding the tree.

Packages within a package distribution file (traditionally with a `.epk` suffix) may be added to the eCos repository or, if using multiple repositories, the top-most repository by clicking on the *Add* button. You will be prompted to provide the location of of the `.epk` package distribution file via a *File Open* dialog box.

Packages may be removed by selecting a package in the tree and then clicking on the *Remove* button. If a package node is selected, all versions of the selected package will be removed. If a package *version* node is selected, only the selected version of the package will be removed.

Using the command line

The **ecosadmin.tcl** script is located in the base of the eCos repository in the `packages` sub-directory. The command operates by default on the eCos repository pointed to by the environment variable `ECOS_REPOSITORY`. Use a command of the following form:

```
$ tclsh ecosadmin.tcl <command>
```

The following commands are available:

add FILE	Adds the packages contained with the specified package distribution file `FILE` to the eCos repository and updates the package database accordingly. This command also takes the op-

tion `--accept_license` which accepts the licensing terms of the packages being installed without displaying them and prompting for their acceptance, as well as `--extract_license` which extracts only license file of the packages into the repository as `pkgadd.txt`.

On windows hosts, this comman includes the option `--noconvert_text` to suppress conversion of text files to windows format (crlf) to retain text files in their original (unix) format. The default is all text files will be converted to native windows format (crlf).

remove PACKAGE [--version=VERSION]	Removes the specified package `PACKAGE` from the eCos repository, as well as all targets which use that package, and updates the package database accordingly. Where the optional version qualifier is used, only the specified version `VERSION` of the package is removed. This command also takes the options `--keep_docs`, which preserves the documentation of removed packages, and `--keep_targets` which retains targets which use the removed package.
list	Produces a list of the packages which are currently installed and their versions. The available templates and hardware targets are also listed.
merge `ECOS_REPOSITORY`	Merges all the packages within the package directory `ECOS_REPOSITORY` into the current repository (defined by the environment variable `ECOS_REPOSITORY`).
check	Check the package database, listing targets that refer to non-hardware packages or non-existent packages, as well as orphaned hardware packages (packages not used by any target).
rmtarget	Remove a target definition from the eCos database file.
clean	Filter out missing packages from the eCos database file.

> **⊗ Warning**
>
> It is possible to remove critical packages such as the common HAL package using this tool. Users should take care to avoid such errors since core eCos packages may only be re-installed in the context of a complete re-installation of eCos.

All commands above take the option `--repository=PATH_TO_REPOSITORY` to provide the path to the repository that is to be operated on. The default is the value of the *ECOS_REPOSITORY* environment variable.

Package Structure

The files in an installed eCos source tree are organized in a natural tree structure, grouping together files which work together into *Packages*. For example, the kernel files are all together in:

```
BASE_DIR/kernel/<version>/include/
BASE_DIR/kernel/<version>/src/
BASE_DIR/kernel/<version>/tests/
```

and µITRON compatibility layer files are in:

```
BASE_DIR/compat/uitron/<version>/include/
BASE_DIR/compat/uitron/<version>/src/
BASE_DIR/compat/uitron/<version>/tests/
```

The feature of these names which is of interest here is the `<version>` near the end. It may seem odd to place a version number deep in the path, rather than having something like `BASE_DIR/<version>/...everything...` or leaving it up to you to choose a different install-place when a new release of the system arrives.

The rationale for this organization is historic as in practice developers maintain their source code under revision control systems such as Mercurial, GIT or SourceSafe using the version directory `current` in place of `<version>` throughout. The version directory `current` is favored by most developers.

The `<version>` schema is still maintained as it does permit developers to easily switch between different versions of specific packages within their configuration without having to manipulate their revision control system or partion packages into their own revision control system. It also permits develops to upgrade only specific packages to newer versions while retaining the remainder on earlier versions that have been previously qualified by a standards or certification agency.

Integration with Revision Control Systems

Many developers have their own source code control system, version control system or equivalent, and will want to use it with eCos and eCosPro sources. Since new releases of eCos and eCosPro come with different pathnames for all the source files, a little work is necessary to import a new release into your source repository.

For example, eCosCentric maintains eCosPro within Mercurial, a distributed source revision control system, using a number of source repositories with all package versions as `current`. An eCosPro release is constructed by merging all repositories into a new single release directory and renaming all `current` directories to `<version>`, as well as setting the `<version>` of all templates.

An eCosPro release may therefore be returned into a source revision control system by renaming all `<version>` directories below `packages` back to `current` and committing the resulting directory structure into the repository.

"current" is recommended because **ecosconfig** recognizes it and places it first in any list of versions.

For example, within a POSIX shell environment such as **bash** (normally available under Linux and provided with the eCosPro host tools for Windows hosts) use the following command:

```
find . -name <version> -type d -printf 'mv %p %h/current\n' | bash
```

Having carried out such a renaming operation, your source tree will now look like this:

```
BASE_DIR/kernel/current/include/
BASE_DIR/kernel/current/src/
BASE_DIR/kernel/current/tests/
...
BASE_DIR/compat/uitron/current/include/
BASE_DIR/compat/uitron/current/src/
BASE_DIR/compat/uitron/current/tests/
```

which is a suitable format for import into your own source code control system. When you get a subsequent release of eCosPro, do the same thing and use your own source code control system to manage the new source base, by importing the new version from

```
NEW_BASE_DIR/kernel/current/include/
```

and so on.

The eCos build tool will now offer only the "current" version of each package; select this for the packages you wish to use.

Alternatively you may choose to create two versions of each package with each `<version>` directory lying alongside the next and `current` containing the most recent `<version>`. For example:

```
cd $ECOS_REPOSITORY
find . -name <version> -type d -printf 'cp -r %p %h/current\n' | bash
```

Having carried out such a copy operation, your source tree will now look like this:

```
BASE_DIR/kernel/current/include/
BASE_DIR/kernel/<version>/include/
BASE_DIR/kernel/current/src/
BASE_DIR/kernel/<version>/src/
```

```
BASE_DIR/kernel/current/tests/
BASE_DIR/kernel/<version>/tests/
...
BASE_DIR/compat/uitron/current/include/
BASE_DIR/compat/uitron/<version>/include/
BASE_DIR/compat/uitron/current/src/
BASE_DIR/compat/uitron/<version>/src/
BASE_DIR/compat/uitron/current/tests/
BASE_DIR/compat/uitron/<version>/tests/
```

In simple terms, while you initially may have two identical version directories, when you overlay a new version on top of an existing directory tree, the older files within current will be replaced by the newer <version>. Within your configuration, however, all packages marked to use the older version will remain fixed while those marked to use current will switch to a newer version.

The above method of overlaying directories however is strongly discouraged. For example, the file ecos.db is not versioned and is common to both releases' packages directories so only the last version of the file will be retained with the contents of previous versions lost. In addition, common target definitions within ecos.db may differ between releases.

To work around these issues, developers are strongly advised to use the **merge** command of the **ecosadmin.tcl** tool which will merge the two versions of ecos.db in addition to overlaying new <version> subdirectories into the target repository. In addition, while **merge** will copy over the newer <version> subdirectories it will not replace the contents of pre-existing subdirectories such as current. This may, however, be made an option in a newer release of the **ecosadmin.tcl** tool and currently can be overcome through the following commands:

```
cd NEW_REPOSITORY
find . -name <version> -type d -printf 'cp -rf %p/. $ECOS_REPOSITORY/%h/current\n' | bash
```

Note

For the example given, the updated target definition in the merged repository will require use of the new device driver not present in the older release as a result of the updated target definition within ecos.db.

Warning

Making such a change has implications for any build trees you already have in use. A configured build tree contains information about the selected packages and their selected versions. Changing the name of the "versioning" folder in the source tree invalidates this information, and in consequence it also invalidates any local configuration options you have set up in this build tree. So if you want to change the version information in the source tree, do it first, before investing any serious time in configuring and building your system. When you create a new build tree to deal with the new source layout, it will contain default settings for all the configuration options, just like the old build tree did before you configured it. You will need to redo that configuration work in the new tree.

Moving source code around also invalidates debugging information in any programs or libraries built from the old tree; these will need to be rebuilt.

Part VI. Fixes and Patches

When an issue has been reported, reproduced and fixed, the fixes to the eCos or eCosPro source code is often provided in the form of a single file known as a *patch* or *patchfile*. This `patchfile` is a text file that contains a list of contexts and differences for a number of different files which represent the changes used to fix the issue.

While the text file can be read and understood by experienced C and C++ programmers, it is more common to provide this file to the patch application which applies those differences to one or more original files, producing patched versions of the files. Normally the patched versions are put in place of the originals. Backups of the originals can also be made.

This section describes the process of applying a provided eCos/eCosPro patch to an eCosPro distribution and provides detailed user instructions.

Table of Contents

Chapter 28. Applying a patch

Patches are normally provided in the form of a single patch file. Save this file to a location from where it can be referenced and loaded by the patch application.

Pre-process patchfile

Patches can be specific to a single version of eCosPro or can be applicable to a number of versions. If a patch is specific to a single version, the `patchfile` provided should already be directed at that specific version and will not require pre-processing, and this section may be skipped. If the patch provided is already version specific, skip forward to the section called "Applying a patch".

If a `patchfile` is intended to be applied against a number of different versions of eCosPro you will need to pre-process the `patchfile` to create a new `patchfile` that is specific to your version. This is because the naming convention of directories used by CDL (see Chapter 22, *CDL Concepts*) includes the version number of individual packages, and `patchfiles` include the path to the files which are to be changed. As the naming convention for versions used is simple and the path names of files can be simply changed within the `patchfile`, a generic `patchfile` directed at the version "`patchfile`" is usually provided rather than providing a `patchfile` for each version of eCosPro affected.

To pre-process the `patchfile`, open a command prompt with the "Shell Environment" shortcut in the eCosPro group and change to the directory that contains the downloaded `patchfile`. Then run the command below replacing *X*, *Y* and *Z* below with the *major*, *minor* and *sub-minor* version numbers of your release respectively. For example, for eCosPro version 3.1.15 the *major* version number is *3*, the *minor* version is *1*, and the *sub-minor* version number is *15*.

Figure 28.1. Pre-process patch on Linux

```
test@ubuntu:~$ cd /tmp/download
test@ubuntu:/tmp/download$ sed 's@/current/@/vX_Y_Z/@' < issue-1001234.pat > issue-1001234-new.pat
test@ubuntu:/tmp/download$
```

Figure 28.2. Pre-process patch on Windows

```
C:\> cd C:\TEMP
C:\TEMP> sed 's@/current/@/vX_Y_Z/@' < issue-1001234.pat > issue-1001234-new.pat
C:\TEMP>
```

For example, use v3_1_15 for version 3.1.15, v3_1_3 for version 3.1.3 and so on.

Figure 28.3. Example of the pre-process of a patch on Linux for version 3.1.15

```
test@ubuntu:~$ cd /tmp/download
test@ubuntu:/tmp/download$ sed 's@/current/@/v3_1_15/@' < issue-1001234.pat > issue-1001234-new.pat
test@ubuntu:/tmp/download$
```

Applying a patch

Application of a patch usually involves the following three steps:

1. Open a Shell
2. Change to a directory within the source code tree
3. Apply the patch

Patch Prefix (-p*n*)

Patches are usually created against the sources of a release, and within a sub-directory of the source tree. As such, the patch prefix can vary in accordance to where in the directory tree the patch was created. The patch author will usually provide instructions on

how to apply the patch, which will normally include this directory from which the patch should be applied and the patch prefix (number of directories, separated by slashes '/,, to be stripped from each file name found within the patch).

For example, supposing the file name in the patch file was `/a/packages/kernel/v4_1_1/src/sched/mlqueue.cxx` setting *-p0* gives the entire file name unmodified, *-p1* gives `a/packages/kernel/v4_1_1/src/sched/mlqueue.cxx` without the leading slash, *-p6* gives `sched/mlqueue.cxx`.

Example: Applying a patch on Linux

This example makes the following assumptions:

- the patch provided has been stored in the file `/tmp/download/issue-1001234-new.pat` (replace `/tmp/download/issue-1001234-new.pat` with the path and filename to the downloaded or pre-processed `patchfile`).
- The patch file has been pre-processed if necessary - see the section called "Pre-process patchfile".
- the patch is to be applied within the sub-directory `$ECOS_REPOSITORY`
- the patch will be applied with a patch level of *0*

Open a command prompt window or shell (for example, opened with the "Shell Environment" shortcut in the eCosPro group) and follow these commands:

Figure 28.4. Application of a patch on Linux

```
test@ubuntu:~$ cd $ECOS_REPOSITORY
test@ubuntu:/opt/ecospro/ecos-3.1.19/packages$ patch -p0 < /tmp/download/issue-1001234-new.pat
patching file net/common/current/include/bootp.h
patching file net/common/current/src/bootp_support.c
test@ubuntu:/opt/ecospro/ecos-3.1.19/packages$
```

Example: Applying a patch on Windows

This example makes the following assumptions:

- the patch provided has been stored in the file `C:\TEMP\issue-1001234-new.pat` (replace `C:\TEMP\issue-1001234-new.pat` with the path and filename to the downloaded or pre-processed `patchfile`).
- The patch file has been pre-processed if necessary - see the section called "Pre-process patchfile".
- the patch is to be applied within the sub-directory `%ECOS_REPOSITORY%`
- the patch will be applied with a patch level of *0*

Open a command prompt window or shell (for example, opened with the "Shell Environment" shortcut in the eCosPro group) and follow these commands:

Figure 28.5. Application of a patch on Windows

```
C:\> cd %ECOS_REPOSITORY%
C:\eCosPro\ecos-3.1.19\packages> patch -p0 < C:\TEMP\issue-1001234-new.pat
patching file net/common/current/include/bootp.h
patching file net/common/current/src/bootp_support.c
C:\eCosPro\ecos-3.1.19\packages>
```

> **Note**
>
> On Windows XP in earlier releases of the eCosPro 3.1 host tools the environment variable `%ECOS_REPOSITORY%` will not be recognised as it used forward instead of backward slashes as directory name seperators. To work around this issue, alter the `%ECOS_REPOSITORY%` variable by changing forward to backward slashes.

Figure 28.6. Application of a patch on Windows XP

```
C:\> echo %ECOS_REPOSITORY%
C:/eCosPro/ecos-3.1.19/packages
C:\> set ECOS_REPOSITORY=C:\eCosPro\ecos-3.1.19\packages
C:\> cd %ECOS_REPOSITORY%
C:\eCosPro\ecos-3.1.19\packages> patch -p0 < C:\TEMP\issue-1001234-new.pat
patching file net/common/current/include/bootp.h
patching file net/common/current/src/bootp_support.c
C:\eCosPro\ecos-3.1.19\packages>
```

Chapter 29. Problems applying a patch

All patches provided by eCosCentric should apply cleanly to your specific version of eCosPro. On the rare occassion that a patch does not apply cleanly, here is a list of error messages and how to handle them. If you are still unable to apply the patch or do not see the error message listed here, please report this issue on the eCosCentric Bugzilla website.

Cannot find file

```
can't find file to patch at input line nnn. File to patch:
No file to patch. Skipping patch. n out of n hunks ignored.
```

First make sure that you used the right patch command line, as shown above. If that still did not work, try with a higher n in -pn, up until maybe 6. To try without actually applying the patch, add `--dry-run` to the command line. With this option the patch is not actually applied but you get all the messages as if the patch were really applied. This allows you to determine whether there would be any errors, without having to the files from backup every time.

When you no longer get the error messages about not finding the file to patch, then you have found the right number for -p and you can repeat the command without `--dry-run` to actually apply the patch.

Hunk FAILED

```
Hunk #n FAILED at nnn. n out of n hunks FAILED - saving rejects to file file.rej
```

This means that one or more changes, called hunks, could not be introduced into the file. Occasionally this could be because the patch was emailed or copied into a file and whitespace was either added or removed. Try adding `--ignore-whitespace` to the command line to work around this.

If you still get errors, it is most likely the `patchfile` was created for a version of eCosPro that is substantially different from the version you are using. In this event, please report this issue on the eCosCentric Bugzilla website.

Hunk succeeded

```
Hunk #n succeeded at nnn (offset n lines) and
Hunk #n succeeded at nnn with fuzz n.
```

Usually there is no problem here and the patch could be applied completely. This message usually means that the `patchfile` was originally created for a slightly different version of eCosPro from the one you are using.

On rare occassions patch may think it applied the changes correctly but really did not. If something does not work and you think that the patch may have been applied incorrectly, please report this issue on the eCosCentric Bugzilla website.

Malformed patch

```
malformed patch at line nnn
```

This message means that the `patchfile` is damaged. If you copy-pasted the patch from somewhere, be careful not to modify or damage it. A `patchfile` contains spaces at the start of lines and these often get lost in copy-paste. The safest mechanism is to download the `patchfile` and save it somewhere locally. If the browser does not download the `patchfile` and displays it instead, use `File -> Save` to save a copy locally.

Reversed patch

```
Reversed (or previously applied) patch detected! Skipping patch.
```

```
Reversed (or previously applied) patch detected! Assume -R? [n]
```

This message usually means that a change in the patch is already contained in the file. The most common reason for this is that the version of eCosPro you are using already contains that patch or the patch was already applied. In this event do not apply the patch.

Part VII. Appendixes and Index

Table of Contents

Appendix A. Target Setup

The documentation for your intended target in the eCosPro Reference Manual should contain information on the board support, eCos timings as well as how to set up the hardware for eCos and how to configure and build eCos and, if applicable, RedBoot or BootUp.

Appendix B. Real-time characterization

For a discussion of real-time performance measurement for eCos, see the kernel documentation in the eCos Reference Manual.

As with the target setup descriptions in the previous appendix, this information may be found alongside the target hardware documentation in the eCosPro Reference Manual

Appendix C. GNU General Public License

                        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

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>, 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.