Getting Started with eCos™

Matsushita AM31/AM33
edition

March 2000

Copying terms

Trademarks

Java™, Sun, and Solaris™ are trademarks and registered trademarks of Sun Microsystems, Inc.

SPARC is a registered trademark of SPARC International, Inc.

UNIX™ is a trademark of The Open Group.

Microsoft, Windows NT, Windows 95, Windows 98 and Windows 2000 are registered trademarks of Microsoft Corporation.

Linux is a registered trademark of Linus Torvalds.

Intel is a registered trademark of Intel Corporation.

eCos™ is a trademark of Red Hat, Inc.

Red Hat is a registered trademark of Red Hat, Inc.

300-400-1010049-03

Contents

Foreword

Welcome to the 1.3.1 release of Red Hat eCos(TM) - the Embedded Configurable Operating System.

What’s New in 1.3.1?

In this, the third major public release of eCos, we have added a wealth of new features, enhancements, and have further extended the target platform coverage.

The configuration system has been completely revised and updated. Major new elements include:

- Package management that supports the extension of eCos functionality via third party add-on packages.
- A standardized configuration save file format that is human readable and editable, and compatible between both GUI and command line configuration tools.
- Enhanced web based help and component documentation system integrated into the GUI configuration tool.
- The Component Definition Language (CDL) has been radically revised and has now been implemented as a TCL extension for maximum flexibility. CDL is now fully documented in the Component Writers Guide.
- Template support for straightforward control of multiple configuration elements, which can be used to provide easy access to standard eCos configurations such as a debug stub boot ROM.
- Best of all, the source of the new configuration tools and underlying libCDL technology has been open sourced under the GNU Public License (GPL).

A companion beta version of the eCos TCP/IP stack has been released in conjunction with 1.3.1. The stack is derived from the OpenBSD source base and provides UDP, TCP, ICMP and BOOTP protocol support on an IPv4 standards base. Device driver support for Cirrus Logic EP72xx evaluation boards and Motorola MBX is included. The stack, ethernet core support, and device drivers are all distributed as configurable eCos packages.

A PCI bus support library has also been added that provides generic PCI bus based device initialization, discovery, and configuration. The library has been ported to both the VR4300 DDB-VRC4373 and StrongARM EBSA285 development boards.

New architectures and platforms added in this release include:

- ARM Thumb
- ARM9
- Cirrus Logic CL-PS7111 and EP72xx
- Cogent CMA222 and CMA230 ARM boards
- Hitachi SH3
- Intel StrongARM
- Intel x86 PC
- Matsushita AM33
- Motorola MBX evaluation board
- NEC MIPS VR4300

For further details of all the changes see the NEWS file in the eCos sources.

This is the first release of eCos since the merger of Red Hat and Cygnus Solutions was completed. Red Hat is dedicated to continued enhancement and maintenance of the eCos system. Developers can look forward to upcoming releases that further expand the architectural and board coverage, extend the functionality of the TCP/IP stack, add a Linux version of the GUI configuration tool, and add major new features such as a Linux/Posix compatibility layer based on the upcoming EL/IX standard - see

http://sourceware.cygnus.com/elix/

for more details.

The merger has brought about some minor changes to eCos's Mozilla-derived public license, the most fundamental of which is simply the change of name from Cygnus eCos Public License (CEPL) to Red Hat eCos Public License (RHEPL). The license terms themselves have not changed in any material way other than alterations necessary to accommodate the change in company details.

eCos in a Nutshell

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 Red Hat GNUPro and 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 Red Hat eCos Public License (RHEPL). It also 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, although we do require that you make publicly available any modifications to the eCos code itself. Red Hat of course welcomes 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.

One of the key technological innovations in eCos is our 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; open source contributors; or additional optional components from Red Hat.

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. We provide, without charge, everything necessary for basic embedded applications development.

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 seven different target architectures (ARM, Hitachi SH3, Intel x86, MIPS, Matsushita AM3x, PowerPC and SPARC) including many of the popular variants of these architectures and evaluation boards. Many new ports are in development and will be released as they become available.

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.

To get the most out of eCos you should visit the eCos open source developers site:

http://sourceware.cygnus.com/ecos/

The site is dedicated to the eCos developer community and contains a rich set of resources including news, FAQ, online documentation, installation guide, discussion and announcement mailing lists, online problem report form, and runtime and development tools downloads. We also support anonymous CVS and WEBCVS access to provide you with direct access to the very latest eCos source base. Complementing the open source developers site is an eCos product site, featuring news, press releases, details of our commercial engineering and support services, products, and third party partner offerings. This is located at

http://www.redhat.com/services/ecos/

We have released eCos as open source software because we believe that this is the most effective software development model, and that it provides the greatest benefit to the embedded developer community as a whole. As part of this endeavor, we seek the input and participation of eCos developers in its continuing evolution. Participation can take many forms including:

- providing us with feedback on how eCos might be made more useful to you - by taking part in the ongoing mailing list discussions and by submitting problem reports covering bugs, documentation issues, and missing features
- contributing bug fixes and enhancement patches
- contributing new code including device drivers, board ports, libraries, and other runtime components

Our long term aim is to make eCos a rich and ubiquitous standard infrastructure for the development of deeply embedded applications. This will be achieved in part by Red Hat’s own efforts, but also with the assistance of the eCos developer community cooperating to improve eCos for all. I would like to take this opportunity to extend our thanks to the many eCos developers who have already contributed feedback, ideas, patches, and code that have augmented and improved this release.

On behalf of the eCos team, welcome to the eCos developer community.

Paul Beskeen,
Director of Engineering,
eCos March 2000

1

Documentation Roadmap

Getting Started with eCos

Release Notes

Description of this release.

Installation Guide

Hardware and software installation instructions, including instructions on how to execute some prebuilt tests to verify the installation.

Programming Tutorial

A tutorial that gets you started running programs with **eCos**.

Appendixes

Extra information about the licensing terms for **eCos**.

eCos User's Guide

The eCos Configuration Tool

A description of all features of the Configuration Tool.

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**.

eCos Reference Manual

Preliminaries

An overview of the **eCos** kernel and configurability system.

Kernel APIs

In-depth description of **eCos**'s native C kernel API, the μITRON API, the ISO standard C library, and the **eCos** Hardware Abstraction Layer (HAL). Important considerations are given for programming the **eCos** kernel. The semantics for each kernel function are described, including how they are affected by configuration.

eCos Device Drivers

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

The ISO Standard C and Math Libraries

eCos comes with an implementation of the ISO C library specification. This section gives details about the implementation, lists the few functions that are not yet implemented, and gives a complete reference for configuring the C library.

Part I: Release Notes

eCos 1.3.1 supports the following architectures:

This release of eCos supports the following architectures:

- Matsushita MN10300 (AM31)
- Matsushita AM33
- Linux i386—synthetic Linux target

This release of eCos supports the following target platforms:

- Matsushita MN10300 stdevall (AM31)
- Matsushita STB System reference Board (AM33)
- Linux (i386) - synthetic Linux target

This release of eCos supports the following host operating systems:

- UNIX™—support for UNIX is still “beta”. Redhat Linux™ and Solaris™ are the only tested UNIX variants.

  Microsoft® Windows NT®, Windows 95®, Windows 98®, Windows 2000®—support for Windows 95, 98 and 2000 is still “beta”.

2

Notation and Conventions

Since there are many supported target architectures, notation conventions will be used to avoid repeating instructions which are very similar.

GDB and GCC Command Notation

Cross-development commands like `gcc` and `gdb` will be shown without prefixed information about the platform for which you are cross-compiling. You need to add the necessary prefix before you execute the commands, so instead of typing ‘gcc’ and ‘gdb’ as in the examples, use:

`mn10300-elf-gcc and mn10300-elf-gdb for MN10300`

`i686-pc-linux-gnu-gcc` and `i686-pc-linux-gnu-gdb for` i386

Note that the GCC cross compiler generates executable files with the `.exe` suffix on Windows, but not on UNIX. 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 installing eCos on Windows (usually `C:/Program Files/Red Hat/eCos`) is different from that on UNIX (usually `/usr/local/ecos-v1_3_1`). Since many command line examples in the tutorials use these paths, this default (base) directory will be shown as *BASE_DIR*.

Windows and UNIX have similar file system syntax, but the MS-DOS command interpreter on Windows uses the backslash character (\) as a path separator, while UNIX and POSIX shells (including the Cygwin bash shell for windows) use the forward slash (/).

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

3 Overview of the Release

The Embedded Configurable Operating System (eCos) software is a set of tools and a run-time environment for developing embedded applications. eCos is a configurable, open source system that allows you to build a run-time system that closely matches the needs of your application.

eCos is aimed at embedded software developers using architectures with tight memory requirements, who want a portable framework for developing their applications.

This chapter outlines the features of eCos version 1.3.1. The initial release version was 1.3, and additional 1.3 releases will incorporate an additional number, represented in this manual by “x”. Please note the exact version number of the version that you are using, because it is incorporated in certain file paths.

If you want to start programming eCos immediately, see “Part II: Installation Guide” on page 29 and “Part III: Programming Tutorial” on page 43.

Hardware Abstraction

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

The eCos HAL has now been ported to numerous architectures, and to one synthetic target, Linux i386. Notes on porting the HAL to new platforms are provided in the *eCos Reference Manual*, part: *The eCos Hardware Abstraction Layer*, section: *Kernel porting notes*.

Embedded Kernel

The core of eCos is a full-featured, flexible, and configurable embedded kernel.

The kernel provides, among other features, multi-threading, a choice of schedulers, a full set of synchronization primitives, memory allocation primitives, and thread manipulation functions (see the *eCos Reference Manual* for the full kernel API).

The kernel is designed so that some parts of it can be changed or replaced without affecting other kernel components.

The following is a partial list of kernel features:

- choice of memory allocation algorithm
- choice of scheduling algorithm
- a rich set of synchronization primitives
- timers, counters, and alarms
- interrupt handling
- exception handling
- cache control
- thread support
- kernel support for multi-threaded debugging with GDB
- trace buffers
- infrastructure and instrumentation

The kernel API and configuration are described in the *eCos Reference Manual*.

Configurability

The eCos kernel and other components can be configured in great detail at compile time, avoiding the addition of unwanted code to the library to be linked with your application code. There is no performance penalty for configuration.

Configuration is fine-grained, so that very small details of eCos’ behavior can be tuned by selecting configuration options.

eCos is organized as a component architecture, with a language to describe the constraints between components and individual configuration options. These constraints are necessary to resolve inconsistent configurations, such as disabling the code which handles the real-time clock, while enabling per-thread timers.

The designer of a component or general-purpose library should write configurable code using a component definition language (CDL). Once that has been done there is no additional burden on the end user (i.e. an embedded systems programmer), who will be able to use eCos' graphical **Configuration Tool** to configure the kernel and basic libraries without needing to understand how the configuration infrastructure works.

A tutorial on how to configure eCos is located in "Configuring and Building eCos from Source" on page 47. The **eCos** *User's Guide* has complete information on running the **Configuration Tool** and CDL.

μITRON and Other Operating Systems

eCos' configurability is the key to simulating different operating systems by using compatibility layers on top of eCos' kernel, because the semantics of basic kernel functions can be configured to match the semantics of other operating systems.

The specification for the μITRON operating system has been implemented on top of eCos. μITRON is configured by selecting appropriate options in the kernel (a real-time clock, the `mlqueue` scheduler, and no timeslicing); and writing a thin layer to map the μITRON system calls.

The μITRON port implements the complete μITRON 3.02 "Standard functionality" (level S) specification, as well as some of the "Extended" (level E) functions. The μITRON implementation is described in more detail in the *eCos Reference Manual*.

ISO C Library

The ISO C and math library shipped with eCos was written to be configurable and tightly integrated with the kernel and the HAL.

By carefully selecting configuration options in the C library, you can significantly reduce the size of the final executable image.

Serial Device Drivers

eCos provides serial device drivers for all supported eCos platforms, with the exception of the i386 Linux synthetic target and most simulator platforms. The serial drivers provide an API (documented in the *eCos Reference Manual*) to control serial ports directly. The standard I/O library can be configured to use them as a transport layer.

ROM Monitor Image

eCos ships with a CygMon ROM monitor for the MN10300, TX39, SPARClite, EDB7211, EDB7212 and EP7211 Development Boards. This includes a GDB stub, thus allowing GDB to be used to debug eCos applications on these evaluation boards. In addition to shipping the actual ROM, the image of that ROM is provided in case you need to burn identical copies for additional boards (see “Target Setup” on page 34).

For the AM33 STB and MN10300 stdeval1 and i386 PC, the ROM images include a GDB stub. This allows GDB to connect to the board and download eCos programs.

For the AM33 STB, the ROM image includes a GDB stub that can be installed in the FLASH ROM on the board.

No ROM image is required for the synthetic Linux target.

For the MN10300, TX39 and SPARClite port of CygMon, the source code to it is included as part of the GNUPro package, so that you may recompile it as described in the GNUPro documentation.

Please note that previous releases of CygMon are incompatible with eCos. You must use the version of CygMon that is provided with eCos rather than an older version of CygMon.

Tests and Examples

Test suites are included for every portion of eCos shipped in this release; these are brief programs that test the behavior of most system calls and libraries in eCos. “Test Suites” on page 60 describes how to build and run these test suites.

The last chapters in “Part III: Programming Tutorial” on page 43 give examples that guide you through running eCos applications, starting from a “Hello world” program and then moving on to more complex programs that use additional kernel features.

GNU Tools and their Documentation

Red Hat's GNUPro Toolkit, which includes the GCC and **G++** compilers and the GDB debugger, is needed to build eCos applications. It is bundled with the CDROM distribution of the eCos Developer's Kit, and is also available on the net at `http://sourceware.cygnus.com/ecos/`.

Online HTML versions of the full GNUPro documentation are included with eCos, as well as a specific GNUPro tools reference guide for your hardware architecture, customized for use with eCos. The full GNUPro documentation can also be found on the web at `http://www.redhat.com/support/manuals/gnupro.html`

NOTE The Linux synthetic i386 target is an exception, as there is (currently) no GNUPro manual. However, the GNUPro source archive contains documentation for the tools. This documentation is usually also included as part of a default Red Hat Linux installation, accessible with the **info** program.

eCos Documentation

The eCos documentation set includes *Getting Started with eCos*, the *eCos User's Guide*, the *eCos Reference Manual*, and a *GNUPro Reference Manual* for your specific architecture.

For users of the eCos Net releases, these are available online in HTML format at `http://sourceware.cygnus.com/ecos/`.

4

Package Contents

eCos Net Release

The eCos Net release consists of the archive files for GNUPro and eCos, which are located on the Red Hat eCos web site

`http://sourceware.cygnus.com/ecos/`

The eCos Net release, because it is digitally distributed only, does not provide ROM images for the various development boards. However, the ROM images for the supported hardware platforms are included in the distribution, so you can burn your own ROM/Flash ICs to work with eCos.

HTML versions of the GNUPro and eCos manuals are included in the distribution, and are also available online.

eCos Developers' Kit

If you have a CD distribution of the eCos Developer's Kit, you will find the following items in your package:

- A card to request printed eCos documentation (*Getting Started with eCos*, the *eCos User's Guide*, and the *eCos Reference Manual*), and the complete GNUPro documentation suite, including an eCos-specific reference manual for your architecture.

  With the card you can also request a copy of μ*ITRON 3.0 An Open and Portable Real-Time Operating System for Embedded Systems*, a book by Dr. Ken

Sakamura.

- eCos version 1.3.1 CDROM with source code and precompiled binaries.

MN10300 Package

The MN10300 package contains eCos-specific monitor PROMs for the Matsushita stdeval1 evaluation board. There are no extras for the Matsushita AM33 System Reference Board in the Developers' Kit.

5 System Requirements

Required

- Standard Intel™ architecture PC running Linux (tested on Red Hat Linux distributions 5.0-6.1). Other versions of Red Hat distributions, or Linux distributions from other vendors should work as well. Also, English or Japanese versions of Microsoft Windows NT version 4.0 (service pack 3 or above must be installed), Windows 95, Windows 98, or Windows 2000.

  Sun™ workstation running Solaris 2.5.1 or later for the SPARC™.

  Support for any platform except for Windows NT 4.0 and Solaris 2.5.1 is beta. In particular, it is only possible to rebuild the GNUPro compiler toolchain on Windows NT 4.0, Solaris 2.5.1, and Linux.
- Enough disk space for the installed distribution. The eCos installation process will detail the various components of eCos and the GNUPro toolkit that can be installed, and their disk space requirements.
- 64MB of RAM and a 350MHz or faster Pentium processor.

  If you are downloading the eCos Net Release distribution from Red Hat’s Sourceware site, you will also need space to store that image and to compile GNUPro and eCos from source.

If you will be using the MN10300 stdeval1 board, you will also need:

- One 16550-based serial port on the PC
- A Matsushita MN10300 standard evaluation board with eCos CygMon Debug PROMs installed

- One standard modem (straight connection) serial cable to connect the serial port on the PC to the evaluation board. An optional second serial cable can be used for diagnostic I/O.

If you will be using the AM33 STB system reference board, you will also need:

- One 16550-based serial port on the PC
- A Matsushita AM33 STB System Reference Board, with the ability to download using the JTAG debugger. To enable debugging using GDB, the eCos “GDB stubs ROM” will need to have been programmed into the Flash ROM.
- Connection to the host computer should be made using a null modem RS232 serial cable. A gender changer may also be required.

If you will be using the Linux synthetic target, you will also need:

- An ix86 PC with an installed Linux distribution (tested with Red Hat Linux distributions 5.0 - 6.1).

Recommended

- We recommend that Windows NT users install Internet Explorer 4.0 or later, since this will allow the **Configuration Tool**’s documentation panel to be searchable.
- A Pentium II computer and 64MB or more of RAM are recommended for best build performance.

The system has been tested only in the recommended configuration above, although other configurations are expected to work.

6
Reporting Problems

Reporting bugs and other problems is very important: it allows Red Hat to solve your problem quickly, and improves the eCos product. The effort you make in reporting problems is appreciated.

To submit a problem report, please use the web interface. If you have a CD distribution of the eCos Developer's Kit, you should use the address:

http://support.cygnus.com.

You will need a login name and an ID, provided by your administrator.

If you are using the eCos Net release you should use the address

http://sourceware.cygnus.com/ecos/problemreport.html.

Known Bugs in eCos and GNUPro

Before filing and bug reports, however, please read the README provided with this release. It describes known problems and possible workarounds in eCos or with the **GNUPro Toolkit**. The file is at the base of the distribution.

How to Report Problems

For documentation discussing the means to report on, edit and query problems, see the following *Accessing Red Hat Web Support to report problems*, or *Additional options* in this chapter.

This documentation serves only as a guide and it is not meant to supercede the Help documentation on the Web Support site. We have tried to make our software as trouble-free as possible. If you do encounter problems, we'd like to diagnose and fix the problem as quickly as possible.

Accessing Red Hat Web Support to Report Problems

If you have a CD distribution, use the following instructions to access the Red Hat Support website.

- Use the following URL in your web browser's address or location dialog box.

  http://support.cygnus.com
- Click on the Case Management System icon, enter your ID and password, and the Welcome page will be displayed.

  Access the Welcome page at any time by using the `Welcome` link (in the navigation bar on the left side of each Web Support page).

  If you have the CD distribution, your details will have been entered in the database, and will be displayed on the Welcome page. If you wish to alter these details, select the Profile link in the navigation bar on the left side of the page.
- Use the links included in the navigation bar on the left side of the page to perform any of the following Red Hat Web Support activities.
  - `New Case` (see Submitting a support request, and the Red Hat Support website)
  - `Query Case` (see Additional options, and the Red Hat Support website)
  - `Add Notes` (see Additional options, and the Red Hat Support website)
  - `Find Solutions` (see Additional options, and the Red Hat Support website)
  - `Profile` (see Additional options, and the Red Hat Support website)
  - `Help` documentation see Additional options, and the Red Hat Support website)
  - `Close Case` (see Additional options, and the Red Hat Support website)

Submitting a Support Request

Use the following instructions to submit a support request, once you have a valid ID established.

- Click on `New Case` to create a new reported problem case.

  The New Case page allows you to complete the creation of a new case. If there is more than one site, select the site relating to your problem.
- Click on `Use This Site ID` button to display a list of the relevant products.

- Select a product from the list and then click on the `Create Case for Selected Product` button.

  Each customer has a valid list of parts of Red Hat products for which they can submit problem reports. These components are part of the Web Support database.
- Type a brief description of the case in the `Case Title` field. You can enter up to 80 characters in this field.
- Select a case type from the `Type` drop-down menu that best describes the case.
- Select a customer severity level from the `Severity` drop-down menu that best describes how severe you view this problem.
- Select a case priority level from the `Priority` drop-down menu that best describes the priority of this case to Red Hat.
- Type a complete description of your case in the `Problem Description` field.

  Use the scrollbars to scroll text in this field. You can add up to 30 kilobytes of text in this field.
- Click on the `Create Case` button at the bottom of the page to create the case in the Red Hat Web Support database. Alternatively, clear the input fields on the New Case page, using the `Clear` button.

After you create your case, the Case Details page displays, which includes the Case ID number that the support database assigns to your case.

To create a new case for a different site and/or part, click the `New Case` link in the navigation bar; then use the previous instructions.

Additional Options

The following documentation discusses the other features for the Red Hat Web Support site. Red Hat has a database to help in determining when problems developed, tracking the problems case from their first report through analysis and resolution. The database can also be used for correlation with other products as well as to other related problems.

- Click on `Query Case` to find an existing problem case in our database.

  You may examine problem cases in the Red Hat Web Support database, searching by solution ID or by entering keywords and/or a key phrase. There are options on this page enabling you to control how your search works.

  At this point, view a problem case’s details, check its status, add notes or close a problem.
- Click on `Add Notes` to add additional data to an existing case in our database.

- Click on `Find Solutions` to search for problem solutions in the database. The search will provide a list of the current problem cases in the Red Hat Web Support database.
- Click on `Profile` to change your profile information and/or your Web Support password in our database. A Profile page wil be displayed.
- Click on `Help` for questions about using the Web Support page. The online help documentation for the Web Support site supercedes this guide; it is not meant to supercede the more updated Help documentation for the Web Support site.
- Click on `Close Case` link to close a case. Closing a case brings the problem to its resolution.

Updating your profile

Clicking on Profile allows you to enter the following details.

- Your contact name
- The primary phone number where Red Hat Support can contact you
- FAX number Red Hat Support can use to send you information
- Your e-mail address
- Your site ID, used to identify your primary site in the Web Support database (a Red Hat representative will provide this information)
- Your site name

Part II: Installation Guide

7

Software Installation

Software Installation on Windows

If you have a CD distribution of the eCos Developer's Kit, you have received the eCos software and its supporting utilities on a single CDROM for installation on a PC-compatible computer running Windows NT 4.0, Windows 95, or Windows 98. If you use NT you must apply the NT 4.0 Service Pack 3 or above before installing eCos. Support is only for Windows NT 4.0. Installations on other Windows platforms are beta.

The following components are provided on the eCos CDROM:

- eCos source code
- Prebuilt eCos libraries and tests
- eCos documentation
- Red Hat GNUPro compiler toolchain for eCos source code compilation
- Red Hat Cygwin environment: this product provides a POSIX compatibility layer on top of Windows NT, and supports the GNUPro tools on Windows NT.
- The GNU user tools—a collection of utilities that developers, particularly those with a UNIX background, will find useful. However, they are not supported by Red Hat.
- Documentation for the GNUPro tools, including a Reference Manual for the particular evaluation board being used to run eCos.

If you have obtained the Net release of eCos for Windows, you will have the distribution in a self-extracting archive. Apart from the difference in medium, the installation procedure for eCos itself will be the same as for the CDROM-based distribution.

The software installation process involves a number of installation utilities. Some familiarity with Windows is assumed.

1. Invoke the file Setup.exe on the CDROM. This will start the installation procedure. If you have the `autorun` feature enabled, Windows will run Setup.exe automatically when the CDROM is inserted into the drive.
2. The setup program will offer to install the GNU user tools. Click `Ok`.
3. You will be prompted for a path in which to install the GNU user tools. The default will be in the `/cygnus/gnupro/i686-cygwin32/i686-cygwin32` hierarchy (usually on drive C). It will then offer to install the source code and documentation for the GNU user tools. It is recommended that you install the documentation, but not the source code, unless you are interested in modifying or recompiling the GNU user tools.
4. At this point the setup program will begin installing eCos. Click `Ok`.
5. The default path offered for eCos installation will be in the `/Program Files/Red Hat` hierarchy (usually on drive C). You may change this path, and indeed you will need to change it if that partition does not have sufficient free disk space available. It is recommended that you accept the default selection of software components for installation.
6. You will be asked to select the program folder under which the eCos menu items will be placed. The default folder name is `Red Hat eCos`.
7. The installation should finish normally, offering to show you the `README` file that contains any last minute information and a list of known problems detected after this document was printed. Once the installation is finished, you can start eCos or view the online documentation by selecting **Start** -> **Programs** -> **Red Hat eCos**, and then choosing an option within this folder, e.g **Configuration Tool**, **Package Administration Tool**, etc.

At this point you are ready to configure and build a customized eCos kernel as described in “Configuring and Building eCos from Source” on page 47.

Software Installation on UNIX

Installation and build instructions for the eCos Net release are available on the Red Hat eCos web site `http://sourceware.cygnus.com/ecos/`

Users of the eCos Developer's Kit under UNIX should use the following instructions, which assume that the CD-ROM is available at `/cdrom/cdrom0`.

1. Extract the eCos repository from the compressed tar archive `ecos-131.taz`, located in the root directory of the CD-ROM using the following commands:

```
# mkdir /usr/local
# cd /usr/local
# zcat < /cdrom/cdrom0/ecos-131.taz | tar xvf -
```

On completion, the eCos repository may be found in the directory `/usr/local/ecos-1.3.1`.

2. Extract the eCos development tools from the compressed tar archive `tool-bin.taz`, located in the root directory of the CD-ROM, using the following commands:

```
# mkdir /usr/cygnus
# cd /usr/cygnus
# zcat < /cdrom/cdrom0/tool-bin.taz | tar xvf -
```

On completion, the executable files of the eCos development tools may be found in the directory `/usr/cygnus/ecos-DEVTOOLSVERSION/H-host-triplet/bin`. The source code for the development tools may optionally be installed in the same way:

```
$ zcat < /cdrom/cdrom0/tool-src.taz | tar xvf -
```

3. Add the eCos development tools and any native tools supporting the eCos build process to the front of your path. Under Solaris, for example, you should modify the PATH environment variable as follows.

Using sh, ksh, or bash:

```
$ PATH=/usr/cygnus/ecos-99r1-991015/H-sparc-sun-solaris2.5/bin:/
usr/xpg4/bin:/usr/ucb:$PATH
$ export PATH
```

Using csh or tcsh:

Note that csh also requires the shell command "rehash" after modifying the path for the path change to take effect.

```
% setenv PATH  /usr/cygnus/ecos-99r1-991015/H-sparc-sun-solaris2.5/
bin:/usr/xpg4/bin:/usr/ucb:$PATH
```

At this point you are ready to configure and build a customized eCos kernel as shown in “Configuring and Building eCos from Source” on page 47.

NOTE The order of directories in the PATH is very important, and build failures may result if the PATH is not set correctly. If you are having difficulties in building eCos, please make sure you have set the PATH exactly as above.

8

Target Setup

Connecting To A Target Via Serial

While eCos supports a variety of targets, communication with the targets happens in one of four ways. These are descibed in general below.

The descriptions are followed by descriptions of each target, providing specific details of how to set up the target (if hardware) and the necessary communication information (such as baud rate for hardware targets, or special connection options for simulator targets).

Most targets will have eCos GDB stubs or CygMon installed. These normally wait for GDB to connect at 38400 baud, using 8 data bit, no parity bit and 1 stop-bit (no hardware flow control). Check the section for your target to ensure it uses this speed. If not, adjust the following instructions accordingly.

The following instructions depend on you to select the appropriate serial port on the host - the serial port which connects to the target's (primary) serial port. On Linux this could be /dev/ttyS0, while the same port on Windows would be named COM1, or /dev/ttya on Solaris. Substitute the proper serial port name in the below.

Connect to the target by issuing the following commands in GDB console mode:

```
(gdb) set remotebaud 38400
(gdb) set mips-force-32bit-saved-gpregs
(gdb) target remote /dev/ttyS0
```

In Insight, connect by opening the **File->Target Settings**... window and enter:

```
Target: Remote/Serial
Baud Rate: 38400
Port: /dev/ttyS0
```

You will also need to open the GDB console window with View->Console and enter “set mips-force-32bit-saved-gpregs” at the prompt

Set other options according to preference, close the window and select **Run->Connect to target**.

Connecting To A Target Via Ethernet

Some targets allow GDB to connect via Ethernet - if so, it will be mentioned in the section describing the target. Substitute the target’s assigned IP address or hostname for <hostname> in the following. The <port> is the TCP port which the eCos GDB stub or **CygWin** is listening on. It is also listed in the section describing the target.

Connect to the target by issuing the following command in GDB console mode:

```
(gdb) target remote <hostname>:<port>
```

In Insight, connect by opening the **File->Target Settings**... window and enter:

```
Target: Remote/TCP
Hostname: <hostname>
Port: <port>
```

Set other options according to preference, close the window and select **Run->Connect to target**.

Connecting To A Simulator Target

GDB connects to all simulator targets using the same basic command, although each simulator may require additional options. These are listed in the section describing the target, and should be used when connecting.

Connect to the target by issuing the following command in GDB console mode:

```
(gdb) target sim [target specific options]
```

In Insight, connect by opening the **File->Target Settings**... window and enter:

```
Target: Simulator
Options: [target specific options]
```

Set other options according to preference, close the window and select **Run->Connect to target**.

Connecting To A Synthetic Target

Synthetic targets are special in that the built tests and applications actually run as native applications on the host. This means that there is no target to connect to - the test or application can be run directly from the GDB console using:

```
(gdb) run
```

or from Insight by pressing the **Run** icon. There is therefore no need to connect to the target or download the application, so you should ignore GDB “target” and “load” commands in any instructions found in other places in the documentation.

MN10300 stdeval1 Hardware Setup

The eCos Developer’s Kit package comes with a pair of EPROMs which provide GDB support for the Matsushita MN10300 (AM31) series evaluation board using CygMon, the Cygnus ROM monitor. Images of these EPROMs are also provided at `BASE_DIR/loaders/mn10300-stdeval1/cygmon.bin`. The LSB EPROM (LROM) is installed to socket IC8 on the board and the MSB EPROM (UROM) is installed to socket IC9. Attention should be paid to the correct orientation of these EPROMs during installation.

The CygMon stubs allows communication with GDB by way of the serial port at connector CN2. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit, and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a standard RS232C serial cable (not a null modem cable). A gender changer may also be required.

MN10300 Architectural Simulator Setup

The MN10300 simulator is an architectural simulator for the Matsushita MN10300 that implements all features of the microprocessor necessary to run eCos. The current implementation provides accurate simulation of the instruction set, interrupt controller, timers, and serial I/O.

In this release, you can run the same eCos binaries in the simulator that can run on target hardware, if built for ROM start-up, with the exception of those that use the watchdog timer.

However, note that AM33 devices required to run eCos are not simulated; therefore you cannot run eCos binaries built for the AM33 under the simulator. For the AM33, the simulator is effectively an instruction-set only simulator.

To simplify connection to the simulator, you are advised to create a GDB macro by putting the following code in your personal GDB start-up file (gdb.ini on Windows and .gdbinit on UNIX).

```
define msim
 target sim --board=stdeval1 --memory-region 0x34004000,0x8
 rbreak cyg_test_exit
 rbreak cyg_assert_fail
end
```

You can then connect to the simulator by invoking the command `msim` on the command line:

```
(gdb) msim
```

You can achieve the same effect by typing out the macro's content on the command line if necessary.

AM33 STB Hardware Setup

The Matsushita AM33 STB System Reference Board may be used in two modes: via a JTAG debugger, or by means of a GDB stub ROM.

Use with GDB Stub ROM

The eCos Developer's Kit package comes with a ROM image which provides GDB support for the Matsushita(R) AM33 STB System Reference Board. To install the GDB stub ROM requires the use of the JTAG debugger and the Flash ROM programming code available from Matsushita. An image of this ROM is also provided at `loaders/am33-stb/gdbload.bin` under the root of your eCos installation.

Ensure that there is a Flash ROM card in MAIN MEMORY SLOT <0>. Follow the directions for programming a Flash ROM supplied with the programming software.

The final programming of the ROM will need to be done with a command similar to the following:

```
fdown "gdbload.bin",0x80000000,16,1
```

Once the ROM has been programmed, close down the JTAG debugger, turn the STB off, and disconnect the JTAG cable. Ensure that the hardware switches are in the following configuration:

```
U U D D D U D D

D = lower part of rocker switch pushed in
U = upper part of rocker switch pushed in
```

This is also the configuration required by the Flash programming code, so it should not be necessary to change these.

Restart the STB and the stub ROM will now be able to communicate with GDB**.** eCos programs should be built with RAM startup.

Programs can then be downloaded via a standard RS232 null modem serial cable connected to the SERIAL1 connector on the STB front panel (the AM33's serial port 0). This line is programmed to run at 38400 baud, 8 data bits, no parity and 1 stop bit (8-N-1) with no flow control. A gender changer may also be required. Diagnostic output will be output to GDB using the same connection.

This procedure also applies for programming ROM startup eCos programs into ROM, given a binary format image of the program from `mn10300-elf-objcopy`.

Use with the JTAG debugger

To use eCos from the JTAG debugger, executables must be built with ROM startup and then downloaded via the JTAG debugger. For this to work there must be an SDRAM memory card in SUB MEMORY SLOT <0> and the hardware switches on the front panel set to the following:

```
D U D D D U D D

D = lower part of rocker switch pushed in
U = upper part of rocker switch pushed in
```

Connect the JTAG unit and run the debugger as described in the documentation that comes with it.

eCos executables should be renamed to have a ".out" extension and may then be loaded using the debugger's "l" or "lp" commands.

Diagnostic output generated by the program will be sent out of the AM33's serial port 0 which is connected to the SERIAL1 connector on the STB front panel. This line is programmed to run at 38400 baud, 8 data bits, no parity, and one stop bit (8-N-1) with no flow control. Connection to the host computer should be using a standard RS232 null modem serial cable. A gender changer may also be required.

Building the GDB stub ROM image

eCos comes with a pre-built GDB stub ROM image for the AM33-STB platform. This can be found at `loaders/am33-stb/gdbload.bin` relative to the eCos installation directory.

If necessary, the ROM image can be re-built as follows:

1. On Windows hosts, open a Bash session using **Start->Programs->Cygnus eCos->eCos Development Environment**

2. Create a build directory and cd into it
3. Run (all as one line):
```
cygtclsh80 BASE_DIR/packages/pkgconf.tcl                        \
  --target=mn10300_am33 --platform stb --startup rom            \
  --disable-kernel --disable-uitron --disable-libc --disable-libm \
  --disable-io --disable-io_serial --disable-wallclock --disable-
watchdog
```

where BASE_DIR is the path to the eCos installation directory.

4. Edit the configuration file `pkgconf/hal.h` in the build directory tree by ensuring the following configuration options are set as follows:
```
#define CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS
#define CYGDBG_HAL_DEBUG_GDB_BREAK_SUPPORT
#undef  CYGDBG_HAL_DEBUG_GDB_CTRLC_SUPPORT
#define CYGDBG_HAL_DEBUG_GDB_THREAD_SUPPORT
#define CYG_HAL_ROM_MONITOR
```
5. Run: `make`
6. Run: `make -C hal/common/current/current/src/stubrom`
7. The file `hal/common/current/src/stubrom` will be an ELF format executable of the ROM image. Use `mn10300-elf-objcopy` to convert this to the appropriate format for loading into the Matsushita flash ROM programmer, mode “binary” in this case:
```
mn10300-elf-objcopy -O binary hal/common/current/src/stubrom/
stubrom stubrom.img
```

i386/Linux Synthetic Target Setup

When building for the synthetic Linux target, the resulting binaries are native Linux applications with the HAL providing suitable bindings between the eCos kernel and the Linux kernel.

NOTE: Please be aware that the current implementation of the Linux synthetic target does not allow thread-aware debugging.

These Linux applications cannot be run on a Windows system. However, it is possible to write a similar HAL emulation for the Windows kernel if such a testing target is desired.

Tools

For the synthetic target, eCos relies on features not available in native compilers earlier than gcc-2.95.1. It also requires version 2.9.5 or later of the GNU linker. If you have gcc-2.95.1 or later and ld version 2.9.5 or later, then you do not need to build new tools. eCos does not support earlier versions. You can check the compiler version using `gcc -v` or `egcs -v`, and the linker version using `ld -v`.

If you have native tools that are sufficiently recent for use with eCos, you should be aware that by default eCos assumes that the tools i686-pc-linux-gnu-gcc, i686-pc-linux-gnu-ar, i686-pc-linux-gnu-ld, and i686-pc-linux-gnu-objcopy are on your system and are the correct versions for use with eCos. But instead, you can tell eCos to use your native tools by editting the configuration value "Global command prefix" (CYGBLD_GLOBAL_COMMAND_PREFIX) in your eCos configuration. If left empty (i.e. set to the empty string) eCos will use your native tools when building.

If you have any difficulties, it is almost certainly easiest overall to rebuild the tools as described on:

http://sourceware.cygnus.com/ecos/getstart.html

9

Running Applications on the Target

To verify both that a hardware target is properly set up, and that the GDB commands used to connect to the target (hardware, simulator or synthetic) work properly on your system, you will now be guided through “downloading” and executing a prebuilt eCos test. The procedure is exactly the same when you want to download and run applications or tests that you have built yourself.

On Windows you must have the bash command line interpreter running with some environment variables which are useful for eCos work. If you have purchased the **eCos Developer’s Kit**, you can select this by selecting **Start->Programs->Red Hat eCos->eCos Development Environment**. If you are using the eCos Net release, you should set the environment variables as shown in the *GNUPro Toolkit Reference Manual*. On Linux, simply open a new shell window.

You will need to change directory to the prebuilt tests that are provided in the eCos installation. Change directory as follows:

```
for the AM31 simulator target:
  $ cd BASE_DIR/prebuilt/am31_sim/tests/kernel/v1_3_1/tests
for the AM33-based Matsushita STB board:
  $ cd BASE_DIR/prebuilt/stb/tests/kernel/v1_3_1/tests
for the AM31-based Matsushita stdeval1 board:
  $ cd BASE_DIR/prebuilt/stdeval1/tests/kernel/v1_3_1/tests
for the i386-based Linux synthetic target:
  $ cd BASE_DIR/prebuilt/linux/tests/kernel/v1_3_1/tests
```

To execute the `thread_gdb` test case on the desired target, run GDB in command line mode using the following command, remembering to substitute the appropriate name for the architecture's gdb:

```
$ gdb -nw thread_gdb
```

GDB will display a copyright banner and then display a prompt (gdb). Connect to the target according to the instructions given earlier (in "Target Setup" on page 34) - via serial or ethernet to hardware targets, or directly for simulator and synthetic targets.

Depending on the target type, you will be notified about a successful connection, and possibly see some output informing you of the current program counter of the target.

Now download the test - effectively loading the test case executable into the memory of the target - by typing this command:

```
(gdb) load
```

Again, depending on the target, you may see some output describing how much data was downloaded, and at what speed. Next, start the test case running. For hardware targets this is done with the 'continue' command, while 'run' must be used on simulators and synthetic targets:

```
(gdb) continue
```

or

```
(gdb) run
```

You should now see a number of text messages appear, such as:

```
PASS:<GDB Thread test OK>
EXIT:<done>
```

NOTE eCos has no concept of the application exiting. All eCos test cases complete and then run in a continuous tight loop. To return control to GDB you must stop the application.

The usual method of stopping an application is with **Ctrl+C**, but **Ctrl+C** may not work on your platform for the prebuilts. First, make default tests and check that they work the same way as prebuilts, then modify your config to enable GDB stubs (if applicable) and break support, so that a **Ctrl+C** character will interrupt the application.

Another way to stop the application is by means of a breakpoint. Before running the application, breakpoint `cyg_test_exit()` to stop an eCos test case at its end.

The full functionality of GDB is now available to you, including breakpoints and watchpoints. Please consult the GNUPro GDB documentation for further information.

Part III: Programming Tutorial

10

Programming with eCos

The remaining chapters of this document compromise 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 accompanying versions of the GNUPro tools. On Windows you will be using the bash command line interpreter that comes with Cygwin, with the environment variables set as described in the GNUPro documentation.

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 certain undesired funtions, etc.), and semantics (selecting scheduler, mutex behavior, etc.). See “Configuring and Building eCos from Source” on page 47.

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 when the product reaches a testable state.

Integrity check of the eCos configuration

While Red Hat 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 all the eCos tests after the project's eCos configuration has been determined. See "Test Suites" on page 60.

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, part of the application development may be possible to do using simulated or synthetic targets.

There are two primary reasons for doing this:

- It may be possible (to some extent) 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 not having to communicate with a stub via serial. And finally, it also removes the need for manually or automatically resetting the target hardware.

This is possible to do since 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 jury-rig 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. While this is possible to do, and has been done, it is not specifically documented or supported by Red Hat. It may become so later.

Therefore, select a simulator or synthetic target and use it for as long as possible doing 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.

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, as it reduces performance.

It may be a 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.

11

Configuring and Building eCos from Source

This chapter documents the configuration of eCos, using the ARM PID board as an example. The process is the same for any of the other 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).

At the end of the chapter is a section describing special issues for this architecture which may affect the way you should configure eCos for your target.

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. The following table summarizes the ways in which an eCos image can be prepared for different types of download.

Table 1: Configuration for various download methods

Download method	***HAL configuration***
Burn hardware ROM	ROM start-up
Download to ROM emulator	ROM start-up

Download method	*HAL configuration*
Download to board with CygMon or GDB stub ROM	RAM start-up
Download to simulator without CygMon or GDB stub ROM	ROM start-up
Download to simulator with CygMon	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 CygMon (or a GDB stub) in the simulator, as described in the GNUPro documentation

NOTE 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.

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 CygMon or GDB stub ROM support, the combinations listed above that use CygMon or GDB stub ROMs 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.

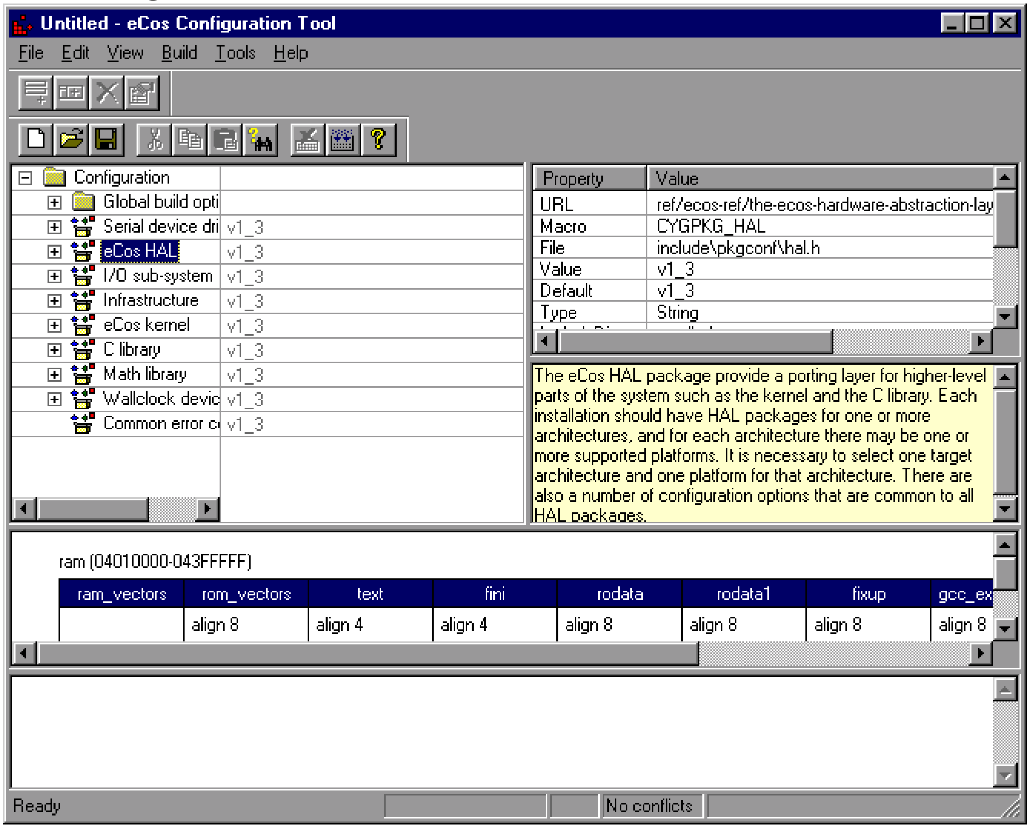
More information on the interactions between CygMon, the simulators, and GDB's thread-aware debugging features is available in the GNUPro Reference Manual for your specific architecture.

Using the Configuration Tool on Windows

Note that the use of the **Configuration Tool** is described in detail in the *eCos User's Guide*.

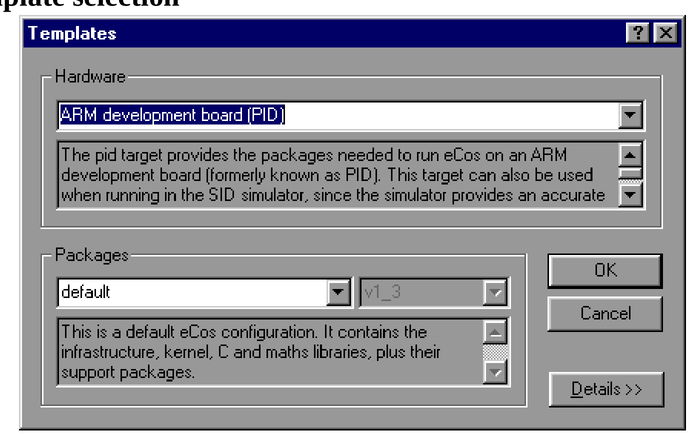
The **Configuration Tool** (see Figure 1) has five main elements: the *configuration window*, the *properties window*, the *short description window*, the *memory layout window*, and the *output window*.

Figure 1: Configuration Tool



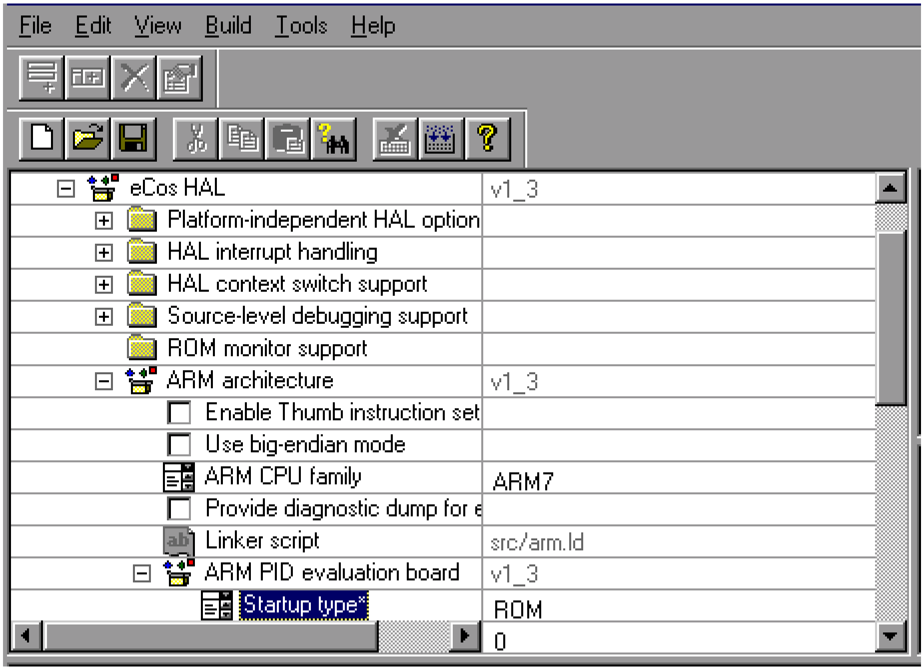
Start by opening the templates window via **Build->Templates**. Select the desired target (see Figure 2).

Figure 2: Template selection



Make sure that the configuration is correct for the target in terms of endianness, CPU model, Startup type, etc. (see Figure 3).

Figure 3: Configuring for the target

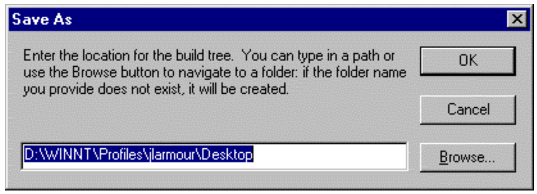


Next, select the **Build->Library** menu item to start building eCos (see Figure 4).

Figure 4: Selecting the build Library menu item

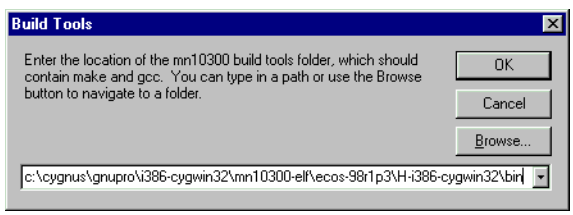
The *Save As* dialog box will appear, asking you to specify a directory in which to place your save file. You can use the default, but it is a good idea to make a subdirectory, called *ecos-work* for example.

Figure 5: **Build dialog**



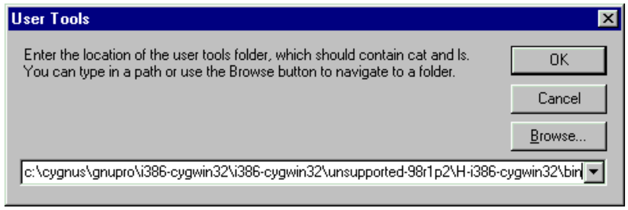
The first time you build an eCos library for a specific architecture, the **Configuration Tool** may prompt you for the location of the appropriate build tools (including make and gcc) using a **Build Tools** dialog box (as shown in Figure 6, page 52). You can select a location from the drop down list, browse to the directory using the **Browse** button, or type in the location of the build tools manually.

Figure 6: **Build tools dialog**



The **Configuration Tool** may also prompt you for the location of the user tools (such as cat and ls) using a User Tools dialog box (as shown in Figure 7, page 53). 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.

Figure 7: **User tools dialog**



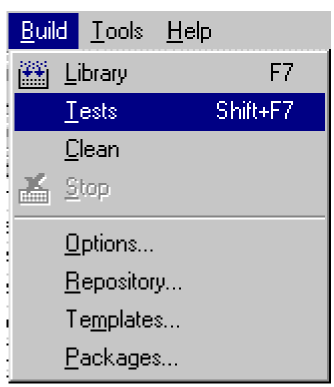
When the tool locations have been entered, the **Configuration Tool** will configure the sources, prepare a build tree, and build the *libtarget.a* library, which contains the eCos kernel and other packages.

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 *libtarget.a*. You should now build the eCos tests for your particular configuration.

You can do this by selecting **Build** -> **Tests**. 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 8: **Selecting the build tests menu item**



“Test Suites” on page 60 will guide you through running one of the test cases you just built on the selected target, using GDB.

Using ecosconfig on UNIX

On UNIX systems the **Configuration Tool** is not yet available, but it is still possible to configure and build a kernel by editing a configuration file manually and using the *ecosconfig* command.

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/ecos-work`. Create this directory and change to it by typing:

```
$ mkdir BASE_DIR/ecos-work
$ cd BASE_DIR/ecos-work
```

It is also necessary to specify the location of the source repository:

```
$ ECOS_REPOSITORY=/opt/ecos/ecos-1.3.1/packages
$ export ECOS_REPOSITORY
```

for sh/ksh/bash users; or

```
% setenv ECOS_REPOSITORY BASE_DIR/packages
```

for csh/tcsh users.

Finally, make sure the tools necessary to build eCos are available from your PATH.

For tools installed with the eCos packages (ecosconfig and ser_filter) - the default RPM installation path is shown - replace as necessary:

```
$ PATH=/opt/ecos/ecos-1.3.1/tools/bin:$PATH
```

For the path for the compiler and debugger tools, the path used in the build instructions is used - replace with the actual path you chose:

```
$ PATH=/install/H-i686-pc-linux-gnu/bin:$PATH
$ export PATH
```

csh/tsch users should do this instead:

```
% set PATH /opt/ecos/ecos-1.3.1/tools/bin:$path
% set PATH /install/H-i686-pc-linux-gnu/bin:$path
```

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.

Table 2: Getting help from ecosconfig

```
$ ecosconfig --help
Usage: ecosconfig [ qualifier ... ] [ command ]
  commands are:
    list                                       : 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)
    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
  qualifiers are:
    --config=FILE                              : the configuration file
    --prefix=DIRECTORY                         : the install prefix
    --srcdir=DIRECTORY                         : the source repository
    --no-resolve                               : disable conflict resolution
    --version                                  : show version and copyright
$
```

Table 3: ecosconfig output — list of available packages, targets and templates

```
$ ecosconfig list
Package CYGPKG_CYGMON (CygMon support via eCos):
 aliases: cygmon
 versions: v1_3_1
Package CYGPKG_DEVICES_WALLCLOCK (Wallclock device code):
 aliases: wallclock devices_wallclock device_wallclock
 versions: v1_3_1
Package CYGPKG_DEVICES_WATCHDOG (Watchdog device code):
 aliases: watchdog devices_watchdog device_watchdog
 versions: v1_3_1
Package CYGPKG_ERROR (Common error code support):
 aliases: error errors
 versions: v1_3_1
Package CYGPKG_HAL (eCos common HAL):
 aliases: hal hal_common
 versions: v1_3_1
Package CYGPKG_HAL_ARM (ARM common HAL):
 aliases: hal_arm arm_hal arm_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_ARM_AEB (ARM evaluation board (AEB-1)):
 aliases: hal_arm_aeb arm_aeb_hal
 versions: v1_3_1
Package CYGPKG_HAL_ARM_CMA230 (Cogent CMA230/222 board):
 aliases: hal_arm_cma230 arm_cma230_hal
 versions: v1_3_1
Package CYGPKG_HAL_ARM_EBSA285 (Intel EBSA285 StrongARM board):
 aliases: hal_arm_ebsa285 arm_ebsa285_hal
 versions: v1_3_1
Package CYGPKG_HAL_ARM_EDB7XXX (Cirrus Logic development board):
 aliases: hal_arm_edb7xxx arm_edb7xxx_hal
 versions: v1_3_1
Package CYGPKG_HAL_ARM_PID (ARM development board (PID)):
 aliases: hal_arm_pid arm_pid_hal
 versions: v1_3_1
Package CYGPKG_HAL_I386 (i386 common HAL):
 aliases: hal_i386 i386_hal i386_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_I386_LINUX (Linux synthetic target):
 aliases: hal_i386_linux
```

```
 versions: v1_3_1
Package CYGPKG_HAL_I386_PC (i386 PC target):
 aliases: hal_i386_pc
 versions: v1_3_1
Package CYGPKG_HAL_MIPS (MIPS common HAL):
 aliases: hal_mips mips_hal mips_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_MIPS_SIM (MIPS simulator):
 aliases: hal_mips_sim mips_sim_hal
 versions: v1_3_1
Package CYGPKG_HAL_MIPS_TX39 (TX39 chip HAL):
 aliases: hal_tx39 tx39_hal tx39_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_MIPS_TX39_JMR3904 (Toshiba JMR-TX3904 board):
 aliases: hal_tx39_jmr3904 tx39_jmr3904_hal
 versions: v1_3_1
Package CYGPKG_HAL_MIPS_VR4300 (VR4300 chip HAL):
 aliases: hal_vr4300 vr4300_hal vr4300_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_MIPS_VR4300_VRC4373 (NEC VRC4373 board):
 aliases: hal_vrc4373 vrc4373_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300 (MN10300 common HAL):
 aliases: hal_mn10300 mn10300_hal mn10300_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300_AM31 (MN10300 AM31 variant HAL):
 aliases: hal_mn10300_am31 mn10300_am31_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300_AM31_SIM (MN10300 simulator):
 aliases: hal_mn10300_sim mn10300_sim_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300_AM31_STDEVAL1 (Matsushita stdeval1 board):
 aliases: hal_mn10300_stdeval1 mn10300_stdeval1_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300_AM33 (MN10300 AM33 variant HAL):
 aliases: hal_mn10300_am33 mn10300_am33_hal
 versions: v1_3_1
Package CYGPKG_HAL_MN10300_AM33_STB (Matsushita STB board):
 aliases: hal_mn10300_am33_stb mn10300_am33_stb_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC (PowerPC common HAL):
 aliases: hal_powerpc powerpc_hal powerpc_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_COGENT (Cogent CMA286/287 board):
 aliases: hal_powerpc_cogent powerpc_cogent_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_FADS (Motorola MPC8xxFADS board):
 aliases: hal_powerpc_fads powerpc_fads_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_MBX (Motorola MBX860/821 board):
 aliases: hal_powerpc_mbx powerpc_mbx_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_MPC8xx (PowerPC 8xx variant HAL):
 aliases: hal_mpc8xx mpc8xx_hal mpc8xx_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_PPC60x (PowerPC 60x variant HAL):
 aliases: hal_ppc60x ppc60x_hal ppc60x_arch_hal
 versions: v1_3_1
Package CYGPKG_HAL_POWERPC_SIM (PowerPC simulator):
```

```
  aliases: hal_powerpc_sim powerpc_sim_hal
  versions: v1_3_1
 Package CYGPKG_HAL_QUICC (Motorola MBX860/821 QUICC support):
  aliases: hal_quicc quicc_hal quicc
  versions: v1_3_1
 Package CYGPKG_HAL_SH (SH common HAL):
  aliases: hal_sh sh_hal sh_arch_hal
  versions: v1_3_1
 Package CYGPKG_HAL_SH_EDK7708 (Hitachi SH7708 board):
  aliases: hal_sh_edk sh_edk_hal
  versions: v1_3_1
 Package CYGPKG_HAL_SPARCLITE (SPARClite common HAL):
  aliases: hal_sparclite sparclite_hal sparclite_arch_hal
  versions: v1_3_1
 Package CYGPKG_HAL_SPARCLITE_SIM (SPARClite simulator):
  aliases: hal_sparclite_sim sparclite_sim_hal
  versions: v1_3_1
 Package CYGPKG_HAL_SPARCLITE_SLEB (Fujitsu MB86800-MA01 board):
  aliases: hal_sparclite_sleb sparclite_sleb_hal
  versions: v1_3_1
 Package CYGPKG_INFRA (Infrastructure):
  aliases: infra
  versions: v1_3_1
 Package CYGPKG_IO (I/O sub-system):
  aliases: io
  versions: v1_3_1
 Package CYGPKG_IO_PCI (PCI configuration library):
  aliases: io_pci
  versions: v1_3_1
 Package CYGPKG_IO_SERIAL (Serial device drivers):
  aliases: serial io_serial
  versions: v1_3_1
 Package CYGPKG_KERNEL (eCos kernel):
  aliases: kernel
  versions: v1_3_1
 Package CYGPKG_LIBC (C library):
  aliases: libc clib clibrary
  versions: v1_3_1
 Package CYGPKG_LIBM (Math library):
  aliases: libm mathlib mathlibrary
  versions: v1_3_1
 Package CYGPKG_UITRON (uITRON compatibility):
  aliases: uitron
  versions: v1_3_1
 Target aeb (ARM evaluation board (AEB-1)):
  aliases: aeb1
 Target am31_sim (MN10300 AM31 minimal simulator):
  aliases:
 Target cma230 (Cogent CMA230/222 board):
  aliases: cma222
 Target cma28x (Cogent CMA286/287 board):
  aliases: cma286 cma287
 Target ebsa285 (Intel EBSA285 StrongARM board):
  aliases: ebsa
 Target edb7xxx (Cirrus Logic development board):
  aliases: edb7211 eb7xxx eb7211
 Target fads (Motorola MPC8xxFADS board):
  aliases:
 Target jmr3904 (Toshiba JMR-TX3904 board):
```

```
 aliases: jmr tx39
Target linux (Linux synthetic target):
 aliases:
Target mbx (Motorola MBX860/821 board):
 aliases: mbx860 mbx821
Target pc (i386 PC target):
 aliases:
Target pid (ARM development board (PID)):
 aliases: PID
Target psim (PowerPC simulator):
 aliases: ppc_sim powerpc_sim
Target sh7708 (Hitachi SH7708 board):
 aliases: edk7708
Target sleb (Fujitsu MB86800-MA01 board):
 aliases:
Target sparclite_sim (SPARClite simulator):
 aliases: sl_sim sparcl_sim
Target stb (Matsushita STB board):
 aliases:
Target stdeval1 (Matsushita stdeval1 board):
 aliases:
Target tx39_sim (TX39 minimal simulator):
 aliases:
Target vrc4373 (NEC VRC4373 board):
 aliases:
Template all:
 versions: v1_3_1
Template cygmon:
 versions: v1_3_1
Template default:
 versions: v1_3_1
Template kernel:
 versions: v1_3_1
Template minimal:
 versions: v1_3_1
Template stubs:
 versions: v1_3_1
Template uitron:
 versions: v1_3_1
```

For detailed information about how to edit the ecos.ecc file, see the *CDL Writer's Guide*.

Selecting a Target

To configure for a listed target, type:

```
$ ecosconfig new <target>
```

For example, to configure for the ARM PID development board, type:

```
$ ecosconfig new pid
```

Then edit the generated file, `ecos.ecc`, setting the options as required for the target (endianess, CPU model, Startup type, etc.)

Create a build tree for the configured target by typing:

```
$ ecosconfig tree
```

You can now run the command *make* or *make tests*, after which you will be at the same point you would be after running the **Configuration Tool** on Windows— you can start developing your own applications, following the steps in “Building and Running Sample Applications” on page 63.

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.

Architectural Notes

There are no notes for this architecture.

12 Test Suites

The eCos kernel and other packages have test suites that rigorously exercise the available features and confirm correct execution. The tests are run on many different possible configurations, but the high number of configuration permutations makes it impossible to test them all. The use of test suites is particularly important for embedded systems, where software robustness is a priority. All eCos software is tested prior to shipping, but if you define your own configuration, you will probably want to verify that the test cases work for it.

This release includes test suites for the eCos kernel, kernel C API, C library, µITRON compatibility, and device driver packages. The use of the test suites is similar for all packages. The tests are supplied as source code for building with your specific eCos configurations. The test case source code is located under the base source directory BASE_DIR/packages/:

- compat/uitron/v1_3_1/tests
- hal/common/v1_3_1/tests
- io/serial/v1_3_1/tests
- devs/wallclock/v1_3_1/tests
- devs/watchdog/v1_3_1/tests
- kernel/v1_3_1/tests
- language/c/libc/v1_3_1/tests
- language/c/libm/v1_3_1/tests

There may be additional tests found in other packages.

Each test suite consists of a number of test cases which can be executed individually. A test case may involve one or more individual tests of the package's features. Successful completion of each test within the test case is reported as a line of text that is sent to the diagnostic channel (usually the serial port) for display on a terminal or terminal emulator.

Each test case runs only once and usually requires target hardware to be reset on completion. Note that certain test cases may not terminate immediately, especially if they involve delays and run on a target simulator.

Using the Configuration Tool

Using the eCos Configuration Tool it is possible to automate the downloading and execution of tests with the appropriately configured eCos packages. To do so, compile and link the test cases by using the **Build->Tests** menu item, after which the tests can be downloaded and executed by selecting **Tools->Run Tests**.

When a test run is invoked, a resizable property sheet is displayed, comprising three tabs: **Executables**, **Output** and **Summary**.

Three buttons appear on the property sheet itself: **Run/Stop**, **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**. This button may be used to interrupt a test run at any point.

See the *eCos User's Guide* for further details.

Using the command line

At the moment, there is no tool for automating testing on Linux, so you will have to run the tests manually.

It may also be necessary to run tests by hand if the automated tool finds any failing tests: it may be necessary to diagnose the problem by debugging the test.

Build the tests by typing 'make tests' in the root of the build directory. This will cause the tests to be built and installed under `<install-path>/tests/`.

Running the test manually is done simply by invoking GDB, connecting to the target, downloading the test, optionally setting some breakpoints, and then running the test. All this was covered in "Target Setup" on page 34.

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. It is the first package to require external testing infrastructure, but it will certainly not be the last.

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 will not need to know about the various testing environments required by the eCos packages, since all contact with the target will continue to happen via GDB.

It remains to be seen if it will be possible, or sensible, to implement all target-external testing infrastructure via filters.

13 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* has two variables you will need to adjust: *PKG_INSTALL_DIR* and *XCC*.

Edit the Makefile, setting *PKG_INSTALL_DIR* to the install tree previously created by `ecosconfig` and uncommenting the relevant *XCC* line for your architecture.

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 instruction (assuming your current working directory is also where you built the eCos kernel):

```
$ gcc -g -IBASE_DIR/ecos-work/install/include hello.c -LBASE_DIR/
ecos-work/install/lib -Ttarget.ld -nostdlib
```

The compilation instruction above contains some standard GCC options (for example, `-g` enables debugging), as well as some mention of paths (`-IBASE_DIR/ecos-work/install/include` allows files like `cyg/kernel/kapi.h` to be found, and `-LBASE_DIR/ecos-work/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 in the simulator using GDB the way you ran the test case. The procedure will be the same, but this time run "gdb" specifying "-nw a.out" on the command line:

```
$ 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 "run" at the (gdb) prompt ("continue" for real hardware) 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 invoke simulator macros, nor 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 `run` at the (**gdb**) prompt) the output should look like this:

```
Starting program: BASE_DIR/examples/twothreads.exe
Entering twothreads' cyg_user_start() function
```

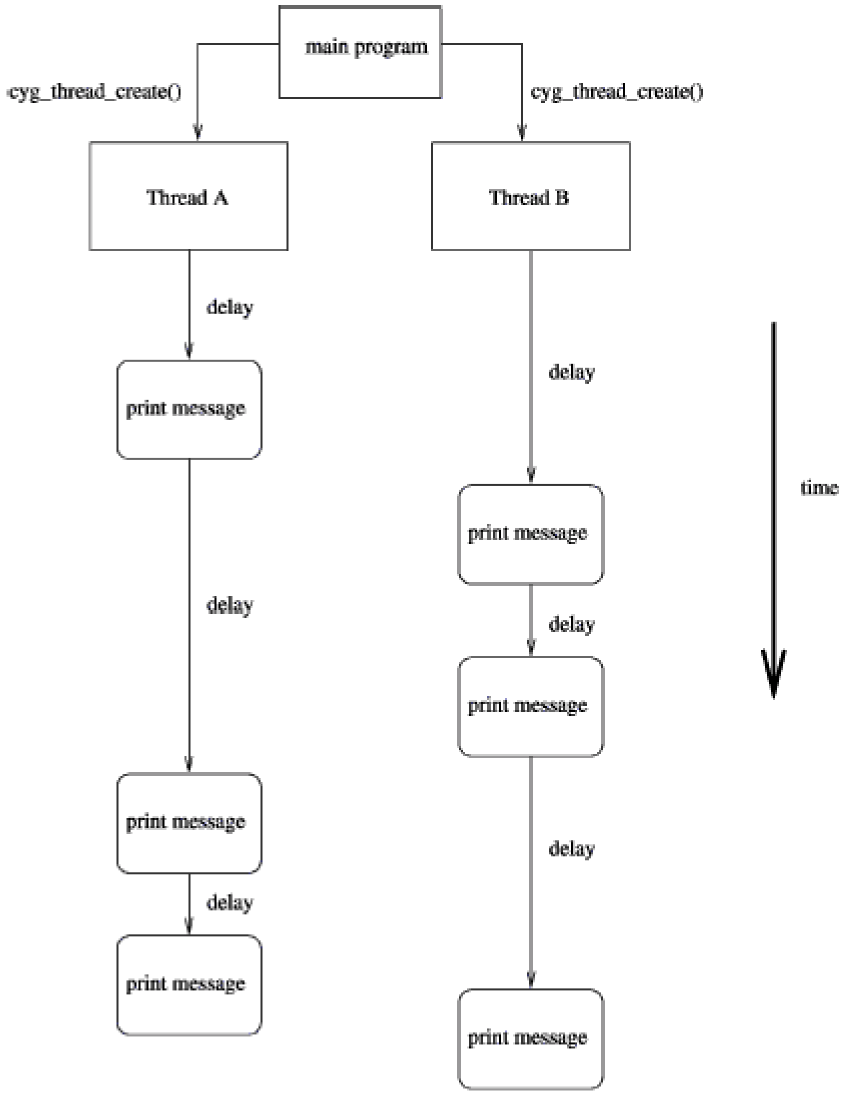
```
Beginning execution; thread data is 0
Beginning execution; thread data is 1
Thread 0: and now a delay of 240 clock ticks
Thread 1: and now a delay of 225 clock ticks
Thread 1: and now a delay of 234 clock ticks
Thread 0: and now a delay of 231 clock ticks
Thread 1: and now a delay of 224 clock ticks
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 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, page 67 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_STDIO_THREAD_SAFE_STREAMS`). Keep in mind that if the C library is thread-safe, you can no longer use **`printf()`** in **`cyg_user_start()`**.

Figure 9: Two threads with simple print statements after random delays



14

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 an infinite 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 interrupted 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.

Table 4: 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 `run` 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 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()`**; this always returns a handle to the default system real-time clock.

- Alarms 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 *Requirements for programs*.
- 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.

Appendixes

Appendix 1: Real-time characterization

For a discussion of real-time performance measurement for eCos, see the *eCos Users' Guide*.

Sample numbers:

Board: Matsushita STDEVAL1 Board

CPU : MN103002A 60MHz

```
eCOS Kernel Timings
Note: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 18 'ticks' overhead
... this value will be factored out of all other measurements
Clock interrupt took   13.73 microseconds (205 raw clock ticks)

Testing parameters:
   Clock samples:          32
   Threads:                24
   Thread switches:       128
   Mutexes:                32
   Mailboxes:              32
   Semaphores:             32
   Scheduler operations:  128
   Counters:               32
   Alarms:                 32


                                 Confidence
     Ave     Min     Max     Var  Ave  Min  Function
  ======  ======  ======  ====== ========== ========
   14.36   11.53   23.53    1.81   54%  33% Create thread
```

```
  2.64    2.53    5.07    0.20   95%  95% Yield thread [all suspended]
  2.25    1.93    4.80    0.31   45%  83% Suspend [suspended] thread
  2.19    2.00    4.93    0.28   91%  91% Resume thread
  3.42    3.00    8.40    0.47   95%  87% Set priority
  0.31    0.13    1.20    0.19   79%  58% Get priority
  8.26    7.40   18.80    0.93   95%  87% Kill [suspended] thread
  2.58    2.47    5.13    0.21   95%  95% Yield [no other] thread
  5.07    4.53    8.67    0.44   62%  50% Resume [suspended low prio] thread
  2.27    2.07    4.53    0.23   87%  87% Resume [runnable low prio] thread
  4.76    4.07    9.40    0.65   66%  75% Suspend [runnable] thread
  2.63    2.53    4.73    0.18   95%  95% Yield [only low prio] thread
  2.09    1.87    4.27    0.27   91%  79% Suspend [runnable->not runnable]
 10.79   10.00   18.20    0.81   95%  79% Kill [runnable] thread
 20.30   18.40   28.80    1.42   79%  54% Resume [high priority] thread
  5.53    5.47   12.13    0.11   98%  97% Thread switch

  0.28    0.27    2.20    0.03   97%  97% Scheduler lock
  1.14    1.13    2.00    0.01   99%  99% Scheduler unlock [0 threads]
  1.14    1.13    2.40    0.02   99%  99% Scheduler unlock [1 suspended]
  1.16    1.13    3.33    0.06   95%  95% Scheduler unlock [many suspended]
  1.23    1.20    3.13    0.05   95%  95% Scheduler unlock [many low prio]

  1.29    1.00    4.20    0.25   65%  50% Init mutex
  2.65    2.47    5.27    0.23   93%  87% Lock [unlocked] mutex
  3.26    3.07    6.80    0.28   93%  87% Unlock [locked] mutex
  2.48    2.33    5.07    0.21   90%  87% Trylock [unlocked] mutex
  2.20    2.07    4.67    0.21   93%  87% Trylock [locked] mutex
  0.23    0.20    1.00    0.05   96%  93% Destroy mutex
 25.11   24.73   27.53    0.21   65%  31% Unlock/Lock mutex

  2.49    2.00    5.73    0.32   81%  37% Create mbox
  0.11    0.00    1.60    0.15   84%  81% Peek [empty] mbox
  3.01    2.60    9.47    0.52   96%  78% Put [first] mbox
  0.10    0.00    1.67    0.15   87%  81% Peek [1 msg] mbox
  3.09    2.60    8.33    0.50   93%  75% Put [second] mbox
  0.06    0.00    1.13    0.08   96%  87% Peek [2 msgs] mbox
  3.10    2.80    7.93    0.40   93%  84% Get [first] mbox
  3.13    2.80    7.53    0.43   90%  78% Get [second] mbox
  2.99    2.60    8.53    0.52   93%  75% Tryput [first] mbox
  2.65    2.33    6.80    0.42   90%  78% Peek item [non-empty] mbox
  3.05    2.73    7.60    0.42   93%  78% Tryget [non-empty] mbox
  3.16    2.93    6.27    0.31   84%  84% Peek item [empty] mbox
  2.48    2.27    5.73    0.30   84%  84% Tryget [empty] mbox
  0.23    0.13    2.07    0.14   96%  87% Waiting to get mbox
  0.22    0.13    1.93    0.13   96%  75% Waiting to put mbox
  3.08    2.80    7.93    0.42   84%  84% Delete mbox
 16.01   15.53   19.00    0.52   78%  59% Put/Get mbox

  0.85    0.67    3.27    0.19   96%  50% Init semaphore
  2.00    1.93    3.87    0.12   96%  90% Post [0] semaphore
  2.05    2.00    3.47    0.09   96%  96% Wait [1] semaphore
  1.85    1.80    3.47    0.10   96%  96% Trywait [0] semaphore
  1.82    1.80    2.53    0.04   96%  96% Trywait [1] semaphore
  0.36    0.33    1.33    0.06   96%  96% Peek semaphore
  0.38    0.33    1.87    0.09   96%  96% Destroy semaphore
 12.38   12.20   16.27    0.30   93%  87% Post/Wait semaphore

  1.18    0.73    4.07    0.24   78%  18% Create counter
  0.20    0.13    1.40    0.11   87%  87% Get counter value
```

```
    0.24    0.20    1.40    0.08   93%  93% Set counter value
    3.17    3.13    4.20    0.07   93%  93% Tick counter
    0.44    0.40    1.73    0.08   96%  96% Delete counter

    2.24    1.67    5.13    0.47   68%  65% Create alarm
    3.86    3.40    9.67    0.51   90%  78% Initialize alarm
    0.15    0.07    1.60    0.12   96%  68% Disable alarm
    3.76    3.47    7.67    0.35   93%  75% Enable alarm
    0.57    0.47    2.73    0.16   96%  84% Delete alarm
    3.64    3.60    4.73    0.07   96%  96% Tick counter [1 alarm]
   21.72   21.67   23.27    0.10   96%  96% Tick counter [many alarms]
    6.13    6.07    8.07    0.12   96%  96% Tick & fire counter [1 alarm]
  101.40   99.53  132.73    2.75   93%  93% Tick & fire counters [>1 together]
   24.21   24.13   26.40    0.14   96%  96% Tick & fire counters [>1 separately]
   11.74   11.60   22.67    0.26   98%  98% Alarm latency [0 threads]
   14.58   11.73   24.93    1.59   54%  28% Alarm latency [2 threads]
   18.18   15.20   41.07    1.96   60%  43% Alarm latency [many threads]

    3.06    2.13   10.33    0.00            Clock/interrupt latency

Timing complete - 23480 ms total

PASS:<Basic timing OK>
EXIT:<done>
```

Appendix 2: eCos Licensing

RED HAT ECOS PUBLIC LICENSE Version 1.1

1. DEFINITIONS.

1.1. “Contributor” means each entity that creates or contributes to the creation of Modifications.

1.2. “Contributor Version” means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. “Covered Code” means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. “Electronic Distribution Mechanism” means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. “Executable” means Covered Code in any form other than Source Code.

1.6. “Initial Developer” means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. “Larger Work” means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. “License” means this document.

1.9. “Modifications” means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. “Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.11. “Source Code” means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or a list of source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor’s choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. “You” means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of fifty percent (50%) or more of the outstanding shares or beneficial ownership of such entity.

1.13. “Red Hat Branded Code” is code that Red Hat distributes and/or permits others to distribute under different terms than the Red Hat eCos Public License. Red Hat’s Branded Code may contain part or all of the Covered Code.

2. SOURCE CODE LICENSE.

2.1. The Initial Developer Grant. The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Initial Developer, to make, have made, use and sell (“Utilize”) the Original Code (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Original Code (or portions thereof) and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

2.2. Contributor Grant. Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Contributor, to Utilize the Contributor Version (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Contributor Version (or portions thereof), and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

3. DISTRIBUTION OBLIGATIONS.

3.1. Application of License. The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code. Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available and to the Initial Developer; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party. You are responsible for notifying the Initial Developer of the Modification and the location of the Source if a contact means is provided. Red Hat will be acting as maintainer of the Source and may provide an Electronic Distribution mechanism for the Modification to be made available. You can contact Red Hat to make the Modification available and to notify the Initial Developer.

3.3. Description of Modifications. You must cause all Covered Code to which you contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims. If You have knowledge that a party claims an intellectual property right in particular functionality or code (or its utilization under this License), you must include a text file with the source code distribution titled “LEGAL” which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If you obtain such knowledge after You make Your Modification available as described in Section 3.2, You shall promptly modify the LEGAL file in all copies You make available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs. If Your Modification is an application programming interface and You own or control patents which are reasonably necessary to implement that API, you must also include this information in the LEGAL file.

3.5. Required Notices. You must duplicate the notice in Exhibit A in each file of the Source Code, and this License in any documentation for the Source Code, where You describe recipients rights relating to Covered Code. If You created one or more Modification(s), You may add your name as a Contributor to the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then you must include such notice in a location (such as a relevant directory file) where a user would be likely to look for such a notice. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions. You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients rights relating to the Covered Code. You may distribute the Executable version of Covered Code under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipients rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer. If you distribute executable versions containing Covered Code, you must reproduce the notice in Exhibit B in the documentation and/or other materials provided with the product.

3.7. Larger Works. You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. INABILITY TO COMPLY DUE TO STATUTE OR REGULATION.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; (b) cite the statute or regulation that prohibits you from adhering to the license; and (c) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it. You must submit this LEGAL file to Red Hat for review, and You will not be able use the covered code in any means until permission is granted from Red Hat to allow for the inability to comply due to statute or regulation.

5. APPLICATION OF THIS LICENSE.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A, and to related Covered Code.

Red Hat may include Covered Code in products without such additional products becoming subject to the terms of this License, and may license such additional products on different terms from those contained in this License. Red Hat may license the Source Code of Red Hat Branded Code without Red Hat Branded Code becoming subject to the terms of this License, and may license Red Hat Branded Code on different terms from those contained in this License. Contact Red Hat for details of alternate licensing terms available.

6. VERSIONS OF THE LICENSE.

6.1. New Versions. Red Hat may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions. Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Red Hat. No one other than Red Hat has the right to modify the terms applicable to Covered Code beyond what is granted under this and subsequent Licenses.

6.3. Derivative Works. If you create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), you must (a) rename Your license so that the phrases “ECOS”, “eCos”, “Red Hat”, “RHEPL” or any confusingly similar phrase do not appear anywhere in your license and (b) otherwise make it clear that your version of the license contains terms which differ from the Red Hat eCos Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH

PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in, the United States of America: (a) unless otherwise agreed in writing, all disputes relating to this License (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration, with the losing party paying all costs of arbitration; (b) any arbitration relating to this Agreement shall be held in Santa Clara County, California, under the auspices of JAMS/EndDispute; and (c) any litigation relating to this Agreement shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

Except in cases where another Contributor has failed to comply with Section 3.4, You are responsible for damages arising, directly or indirectly, out of Your utilization of rights under this License, based on the number of copies of Covered Code you made available, the revenues you received from utilizing such rights, and other relevant factors. You agree to work with affected parties to distribute responsibility on an equitable basis.

13. ADDITIONAL TERMS APPLICABLE TO THE RED HAT ECOS PUBLIC LICENSE.

Nothing in this License shall be interpreted to prohibit Red Hat from licensing under different terms than this License any code which Red Hat otherwise would have a right to license.

Red Hat and logo - This License does not grant any rights to use the trademark Red Hat, the Red Hat logo, eCos logo, even if such marks are included in the Original Code. You may contact Red Hat for permission to display the Red Hat and eCos marks in either the documentation or the Executable version beyond that required in Exhibit B.

Inability to Comply Due to Contractual Obligation - To the extent that Red Hat is limited contractually from making third party code available under this License, Red Hat may choose to integrate such third party code into Covered Code without being required to distribute such third party code in Source Code form, even if such third party code would otherwise be considered “Modifications” under this License.

EXHIBIT A.

The contents of this file are subject to the Red Hat eCos Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.redhat.com

Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is eCos - Embedded Configurable Operating System, released September 30, 1998.

The Initial Developer of the Original Code is Red Hat. Portions created by Red Hat are Copyright (C) 1998, 1999, 2000 Red Hat, Inc. All Rights Reserved.

EXHIBIT B.

Part of the software embedded in this product is eCos - Embedded Configurable Operating System, a trademark of Red Hat. Portions created by Red Hat are Copyright (C) 1998, 1999, 2000 Red Hat, Inc. (http://www.redhat.com) All Rights Reserved.

THE SOFTWARE IN THIS PRODUCT WAS IN PART PROVIDED BY RED HAT AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Appendix 3: The eCos Copyright Assignment Form, Revision 1.1

Rationale

This preamble describes how to use the standard **eCos** copyright assignment form. The rationale behind this assignment is to avoid any possible confusion over the legal ownership of **eCos**, and to indemnify Red Hat and all **eCos** users against copyright or patent claims on contributed code used within **eCos**. Red Hat would be especially vulnerable, but all users and their **eCos** based applications could be affected.

All contributions to **eCos** for which there are copyright assignments will be covered by the Red Hat **eCos** public license. The license provides a guarantee that the contribution will remain freely available to all.

This agreement gives Red Hat ownership of your changes but promises that you will retain the right to use your contributed changes as you see fit.

Because employers often can claim ownership over things that employees write, you may also have to get your employer to sign a disclaimer that says that they have no claim to the changes you are contributing.

Please read everything, and if you have any questions, email

ecos-assign@redhat.com

for help.

Thanks for your contribution to **eCos**!

How to assign copyright

The way to assign copyright to Red Hat is to sign an assignment contract. This is what makes Red Hat the legal copyright holder, so that Red Hat can register the copyright on the new version.

If you are employed as a programmer (even at a university), or have made an agreement with your employer or school that gives them ownership of the software you write, then Red Hat needs a signed letter from your employer disclaiming rights to the contributed software.

The disclaimer should be printed on the company's headed paper, and signed by an officer of the company, or someone authorized to license the company's intellectual property. Here is an example of wording that can be used for this purpose:

<INSERT COMPANY NAME> hereby disclaims all copyright interest in the changes and enhancements made by <INSERT YOUR NAME> to **eCos**, including any future revisions of these changes and enhancements.

<INSERT COMPANY NAME> affirms that it has no other intellectual property interest that would undermine this release, or the use of **eCos**, and will do nothing to undermine it in the future.

<INSERT SIGNATURE OF OFFICER OF COMPANY>,
<INSERT DATE>
<INSERT PRINTED NAME OF OFFICER OF COMPANY>
<INSERT TITLE OF OFFICER>

If your employer says they do have an intellectual property claim that could conflict with the use of the program, then please contact Red Hat to discuss possible next steps.

Below is the usual assignment contract. You need to edit and replace <INSERT NAME OF CONTRIBUTOR> with your full name. Please print a copy, sign, date, and mail it to:

Legal Department (eCos Assignments)
Red Hat, Inc.
2600 Meridian Parkway
NC 27713
USA

Don't forget to include the original signed copy of the employer's disclaimer.

Please try to print the whole first page of the form on a single piece of paper. If it doesn't fit on one printed page, put it on two sides of a single piece of paper, and attach the second page of the form. Please write the date using letters rather than numbers to avoid any confusion due to international day/month ordering conventions.

Note: This text is also available in the **eCos** software distribution, in the file assign.txt.

```
 --------------------------------- Cut Here ----------------------------------

eCos ASSIGNMENT
```

For good and valuable consideration, receipt of which I acknowledge, I, <INSERT NAME OF CONTRIBUTOR>, hereby transfer to Red Hat, Inc. my entire right, title, and interest (including all rights under copyright) in my changes and enhancements to eCos, subject to the conditions below. These changes and enhancements are herein called the “Work”. The Work hereby assigned shall also include any future revisions of these changes and enhancements hereafter made by me.

Upon thirty days prior written notice, Red Hat agrees to grant me non-exclusive rights to use the Work (i.e. just my changes and enhancements, not eCos as a whole) as I see fit; (and Red Hat’s rights shall otherwise continue unchanged).

I hereby agree that if I have or acquire hereafter any patent or interface copyright or other intellectual property interest dominating the software enhanced by the Work (or use of that software), such dominating interest will not be used to undermine the effect of this assignment, i.e. Red Hat and the general public will be licensed to use, in that program and its derivative works, without royalty or limitation, the subject matter of the dominating interest. This license provision will be binding on my heirs, assignees, or other successors to the dominating interest, as well as on me.

I hereby represent and warrant that I am the sole copyright holder for the Work and that I have the right and power to enter into this contract. I hereby indemnify and hold harmless Red Hat, its officers, employees, and agents against any and all claims, actions or damages (including attorney’s reasonable fees) asserted by or paid to any party on account of a breach or alleged breach of the foregoing warranty. I make no other express or implied warranty (including without limitation, in this disclaimer of warranty, any warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE).

Agreed: [signature]

[Print Name]

Date: [Please write using letters]

For Red Hat:

Date:

-------------------- Cut Here and print on separate page ---------------------

[Please print your name here]

[For the copyright registration, of what country are you a citizen?]

[In what year were you born?]

[Please write your email address here]

```
[Please write your address here, so we can mail a signed copy of the agreement
back to you]


[Please write a brief description of the contribution]


[Which files have you changed so far, and which new files have you written so
far?]
```

Index

Symbols