CHAPTER 1

An Introduction to the eCos World

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

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

1.1 Where It All Started—Cygnus Solutions

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

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

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

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

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

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

1.2 The Origins of eCos

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

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

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

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

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

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

1.2.1 In a Word: Configurability

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

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

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

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

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

1.2.2 The eCos Configuration Method

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

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

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

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

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

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

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

Code Listing 1.1 Example code of source-level configuration.

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

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

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

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

1.2.3 eCos Core Components

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

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

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

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

1.2.4 Processor and Evaluation Platform Support

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

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

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

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

1.2.5 eCos Support

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

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

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

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

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

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

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

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

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

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

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

http://bugzilla.redhat.com/bugzilla

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

ecos-maintainers@redhat.com

1.3 Architecture Overview

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

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

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

1.3.1 eCos Terminology

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

1.3.1.1 Component Framework

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

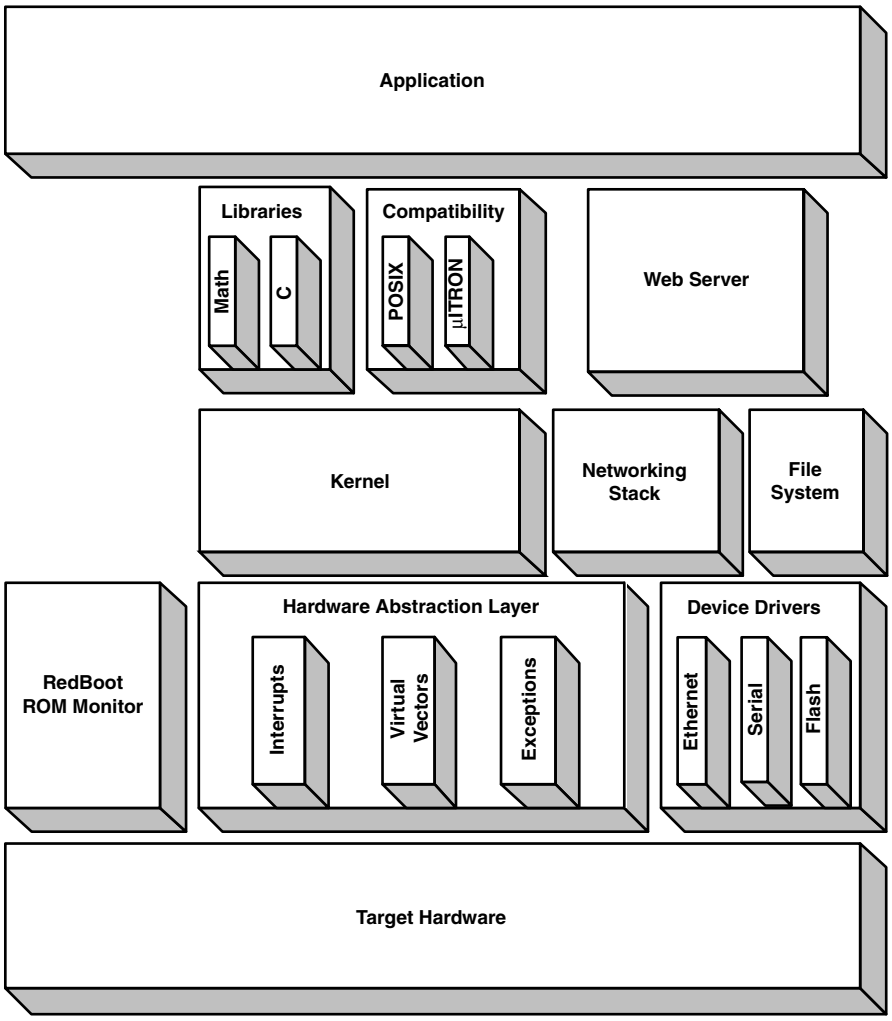

Figure 1.1 Example embedded software system showing layering of eCos packages.

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

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

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

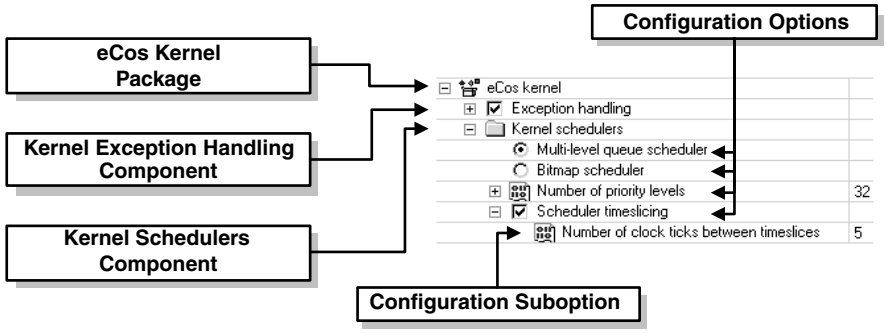

Figure 1.2 Example of the configuration building blocks that compose a package.

1.3.1.2 Component Repository

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

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

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

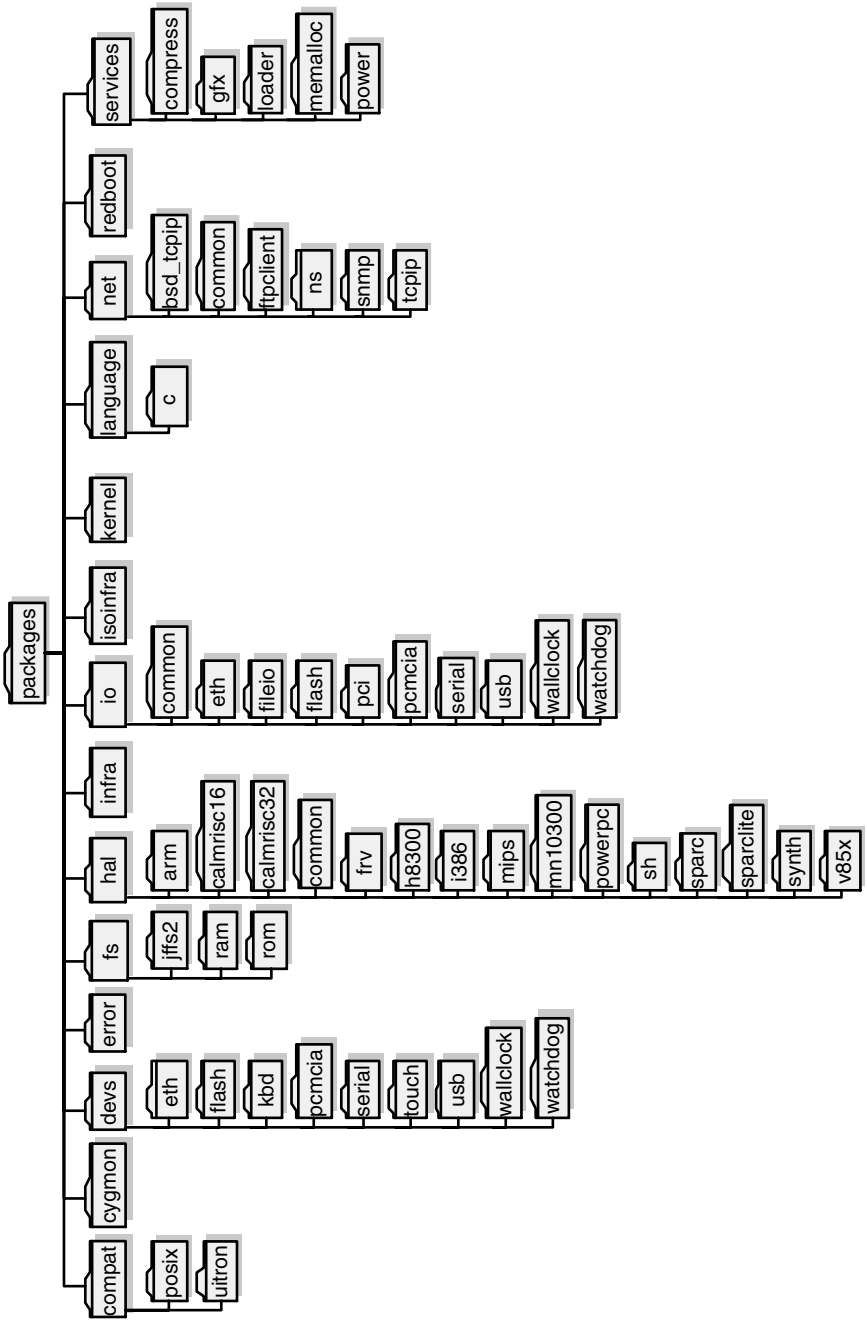

Figure 1.3 High-level component repository directory structure snapshot.

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

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

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

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

Table 1.1 Component Repository Directory Structure Descriptions

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

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

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

[a] The RedBoot ROM monitor has replaced the Cygmon debug monitor.

1.3.1.3 Configuration Options

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

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

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

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

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

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

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

1.3.1.4 Components and Packages

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

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

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

1.3.1.5 Targets

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

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

1.3.1.6 Templates

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

Table 1.2 eCos Templates

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

Table 1.2 eCos Templates *(Continued)*

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

1.4 Summary

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

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