2

Installation Procedure

2.1 Introduction

This chapter describes the process of installing and configuring **RTEMS** and a cross-development environment based on freely available tools and libraries.

2.2 RTEMS FTP Site Organization

RTEMS is distributed only via anonymous ftp.

This section will discuss how to navigate the **RTEMS** ftp site and unarchive the files in the **RTEMS** and GNU package distributions. All example commands will be given in a shell independent fashion unless otherwise noted.

Throughout the rest of this manual **<RTEMS_distribution>** will be used as the parent of components within the **RTEMS** distribution. For persons using the ftp distribution found on the primary ftp site for **RTEMS**, **<RTEMS_distribution>** is **/pub/rtems/releases/3.2.0** on **lancelot.gcs.redstone.army.mil**.

<GNU_distribution> will be used as the parent of the GNU tools available on the primary ftp site for **RTEMS**. **<GNU_distribution>** is **/pub/gnu** on **lancelot.gcs.redstone.army.mil**.

<Modules_distribution> will be used as the parent of the Modules packages available on the primary ftp site for **RTEMS**. **<Modules_distribution>** is **/pub/Modules** on **lancelot.gcs.redstone.army.mil**.

2.2.1 RTEMS Release Directory Organization

The archive files for **RTEMS** Release 3.2.0 are found under the directory **<RTEMS_distribution>**. This directory contains the major subdirectories **ada**, **c**, **tools**, and **patches**. Also contained within this directory are the files INSTALL, LICENSE, and SUPPORT which contain information about the **RTEMS** installation procedure, **RTEMS** executive licensing, and technical support information, respectively. The following sections discuss each of the major subdirectories in the distribution.

2.2.1.1 <RTEMS_distribution>/ada

The complete source code and documentation set for the **Ada** language implementation of **RTEMS** is provided in the **ada** directory. Complete copies of the **Ada RTEMS** source code and documentation are contained within the two archive files, rtems-3.2.0-ada_src.tgz and rtems-3.2.0-ada_doc.tgz. In addition, each manual contained in rtems-3.2.0-ada_doc.tgz is also provided as an individual archive file. The separated manual archive files are found in the subdirectory **individual_manuals.**

2.2.1.2 <RTEMS_distribution>/c

The complete source code and documentation set for the **C** language implementation of **RTEMS** is provided in the **c** directory. Complete copies of the **C RTEMS** source code and documentation are contained within the two archive files, rtems-3.2.0-c_src.tgz and rtems-3.2.0-c_doc.tgz. In addition, each manual contained in rtems-3.2.0-c_doc.tgz is also provided as an individual archive file. The separated manual archive files are found in the subdirectory **individual_manuals.** The **c** directory also contains the source code and formatted documentation for the Cygnus NEWLIB **C** Standard Library. The Cygnus NEWLIB **C** Standard Library source code and documentation are contained within the two archived files, newlib_rtems-050895.tgz and ???_doc.tgz.

2.2.1.3 <RTEMS_distribution>/patches

The **patches** subdirectory contains the patches to the **RTEMS** source code issued since the release of 3.2.0 of **RTEMS**. It is hoped that this directory will be empty.

2.3 Unarchiving the RTEMS and GNU Components

Many of the components of the **RTEMS** release are "tarred, zipped" files and have the **.tar.gz** or **.tgz** extension. The GNU zip package is required to unarchives these

files on the **RTEMS** ftp site. If this package is not installed, the source can be found in **/pub/gnu/gzip/gzip-1.2.4** on **lancelot.gcs.redstone.army.mil**. It may be restored using a command similar to the following:

```
tar xvf gzip-1.2.4.tar
```

This will create a subdirectory **gzip-1.2.4** in the current directory. Please examine the files README and INSTALL and follow the instructions provided there.

[Note: The GNU and tools and Modules package in this distribution use the standard GNU packaging. They will unarchive into a directory based on the package name and version number. For detailed instructions on compilation and installation of the GNU tools and Modules, please refer to those packages.]

Files which have been "tarred, zipped" (i.e. **.tar.gz** or **.tgz** extension) may be unarchived with a command similar to one of the following:

```
gzcat <file>.tgz | tar xvof -
                                OR
gunzip -c <file>.tgz | tar xvof -
                                OR
gtar xzvf <file>.tgz
```

NOTE: *gunzip -c* is equivalent to *gzcat*, while *gtar* is GNU tar.

Given that the necessary utility programs are installed, any of the above commands will extract the contents of **<file>.tar.gz** into the current directory. All of the **RTEMS** components will be extracted into the subdirectory **rtems-3.2.0**. To view the contents of a component without restoring any files, use a command similar to the following:

```
gzcat <file>.tgz | tar tvf -
```

2.4 Installing a Cross-Development GNU Toolset

This sections describes how to build and install the FSF GNU tools for use as a cross-compilation system. These tools are used by the **RTEMS** developers. Every effort has been made to make these instructions accurate and complete. However, it is recommended that the individual doing the installation read the appropriate installation notes for each of the tools in the cross toolset. This will help insure that there are no special requirements for a particular host.

Instructions are provided for building and installing the following tools in the cross-toolset:

- *FSF Binary Utilities*
- *FSF GCC*
- *Cygnus NEWLIB (with **RTEMS** awareness)*
- *FSF GNU Debugger*

- *FSF libg++*

Currently no instructions are provided for the installation of the GNU debugger gdb or libg++.

The Intel GNU/960 tools are installed using a procedure which is different than that used by the FSF GNU tools. It is recommended that the user carefully follow the instructions provided with that toolset. Refer to the section **GNU/960 Notes** for more informations.

If the host and target processors are the same, then it may be possible to use the host development tools. An example of this scenario is using a Hewlett Packard PA-RISC based workstation to develop an **RTEMS** application for the PA-RISC processor. Although the native toolset is useable in this scenario, it is ultimately more desirable to build a toolset specifically for the embedded environment. The main advantage is that it makes it easier to avoid dependencies on the host environment such as accidental use of header files and libraries which are not intended for use anywhere but the development system.

The documentation for the FSF GNU and Cygnus tools is formatted using TeX. The **RTEMS** developers use TeX 3.14t3 to format the manuals for their own use. This document does not contain instructions on the acquisition or installation of TeX and supporting tools.

2.4.1 Assumptions for the Host Development System

This procedure assumes the successful installation of the GNU **C** compiler for use as a native compiler on the development system. On some systems, it may also be necessary to install the GNU Binary Utilities (binutils) to successfully install the GNU **C** compiler. The following packages are assumed to be installed on the development system:

- *gcc 2.6.3*
- *gmake 3.73*
- *binutils 2.5.2 (necessary on some hosts)*
- *gzip 1.2.4*
- *gtar 1.11.2*

[NOTE: It should be possible to use versions of the packages newer than those specified above.]

The documentation for the GNU tools is formatted using TeX. The **RTEMS** developers use TeX 3.14t3 to format the GNU manuals for their own use. This

document does not contain instructions on the acquisition or installation of TeX and supporting tools.

2.4.2 Preparation

Some preparation is required before begining the building and installation of the cross-development toolset.

The first decision which must be made is where the executables for the tools will be installed. This directory should be accessible to all developers and in many environments resides on a server. This directory should be accessed using the same full patch by all developers. This will be referred to as **<install_point>** in the following sections.

When performing cross-development the distinction between host and target environments is critical. The distinction is especially noticeable in the world of embedded systems. Embedded systems are typically under-equipped to be used as a development system. More often than not, the host system cannot directly execute the binaries intended for the target system.

The **configure** script will attempt to guess the proper configuration for the host system if a host configuration is not explicitly provided. The **RTEMS** developers feel that it is best to explicitly provide the host configuration to the **configure** script. The configuration which would be used by default may be obtained by running the **./config.guess** script in the top level diectory of any FSF GNU package. For example, the host configuration guessed for a Sun SparcStation 10 running the Solaris 2.3 operating system is "sparc-sun-solaris2.3." By explicitly obtaining and providing the host information to the **configure** script, this gives the user the opportunity to verify that the host configuration is correct before proceeding. The host configuration will be referred to as **<host_configuration>** in the following sections.

The choice of a target configuration is independent of that of the host configuration with the FSF GNU tools. The target configuration choice reflects the processor in the embedded target. The target configurations used by the **RTEMS** developers are of the form **CPU-OBJECT_FORMAT**, where **CPU** reflects the processor family used on the embedded target and **OBJECT_FORMAT** reflects the desired object format. Object formats supported by the FSF GNU tools vary from processor to processor but typically include a.out, coff, elf, and/or dwarf. If the choice for embedded developers is not constrained by an external tool such as a processor emulator, then the typical order of preference is dwarf, elf, coff, and a.out. The target configuration will be referred to as **<target_configuration>** in the following sections.

After the cross development toolset has been built and installed, it will be necessary to modify the environment of each **RTEMS** application developer to reflect at least the path of the newly installed cross development toolset. For sites which use the Modules package, this will only require the addition of a single new Module file such as m68k-cross which may be loaded by each developer.

2.4.3 Installing FSF Binary Utilities

The FSF Binary Utilities (binutils) package contains the GNU assembler as well as the object file utilities such as a loader (ld), librarian (ar), symbol table extractor (nm), and an object format conversion utility (objcopy).

The following binutils configurations are used by the **RTEMS** project:

Processor Family	Target Configuration
Intel i386	i386-coff
Intel i960	nindy960
Hewlett Packard PA-RISC	hppa-proelf
Motorola m68k	m68k-coff

NOTE: For "UNIX" processors, the native binary utilties should be used.

After unarchiving the binutils distribution (discussed earlier), the binutils package must be configured for the desired host and target. This is accomplished by changing the current directory to the top level of the binutils source tree and executing the **configure** script with the appropriate arguments. This step configures the binutils source so it can be compiled to execute on the specified **<host_configuration>** and produces code compatible with the GNU Assembler and Loader for the **<target_configuration>**. The *-with-targets=all* informs the configuration process that the executables produced by this build should have support for all of the object formats possible. This makes the resulting executables larger but insures that all supported formats are available for output. This avoids rebuilding and reinstalling the binutils package when it is discovered that some external tool such as an emulator only supports a certain object format. The executables will be installed at **<install_point>**. A command sequence similar to the following should be used:

```
cd binutils-<BINUTILS_Version>
CC=gcc ./configure --host=<host_configuration>
--target=<target_configuration> \
    --with-gnu-as --with-gnu-ld --with-targets=all \
    --prefix=<install_point> \
    --exec-prefix=<install_point> \
    --local-prefix=<install_point>
```

If the **configure** script successfully completes, then the GNU Binary Utilities may be built. This step builds the binutils package in the local directory and does **NOT** install any files. The example commands specifies that *gcc* should be used as the **C** compiler and the arguments to be used by *gcc*. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g"
```

If the build process successfully completes, then the binutils package is ready to install. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g" install
```

If this successfully completes, then the binutils package has been installed at **<install_point>**. The executable files are installed with a name which includes the target configuration to insure that the cross tools have distinct names from their native counterparts. For example, *ar* is installed as ***<target_configuration>****-ar*.

The documentation for the binutils package is formatted using TeX. If TeX and some supporting tools are installed on the development system, then the following command may be used to produce the documentation in the "DVI" format:

```
gmake CC="gcc" CFLAGS="-O4 -g" dvi
```

2.4.4 Installing FSF gcc

The FSF GCC package is the GNU **C**, **C++**, and **Objective-C** compilers. This compiler supports a wide variety of host and target systems. Used in conjunction with the Binary Utilities, GNU **C** can be used as a cross compiler to produce executables for a large number of target architectures. It is recommended that version 2.6.3 of GCC be used.

- *As of gcc 2.6.3, the crossgcc patches from ftp.cygnus.com:/pub/embedded were required to support the configurations used internally by the **RTEMS** project. These configurations are supposed to be incorporated into the standard FSF distribution as of gcc v2.7.x. If this is not the case, there will most likely be an updated crossgcc patch on ftp.cygnus.com at that time. If this patch is required, then it **MUST** be installed before configuring GCC.*

The following GCC configurations are used by the **RTEMS** project:

Processor Family	Target Configuration
Intel i386	i386-coff
Intel i960	nindy960
Hewlett Packard PA-RISC	hppa-proelf
Motorola m68k	m68k-coff

NOTE: For "UNIX" processors, the native installation of gcc should be used.

After unarchiving the gcc distribution (discussed earlier), the gcc package must be configured for the desired host and target. This is accomplished by changing the current directory to the top level of the gcc source tree and executing the **configure** script with the appropriate arguments. This step configures the gcc source so it can be compiled to execute on the specified **<host_configuration>** and produces code compatible with the GNU Assembler and Loader for the **<target_configuration>**. The executables will be installed at **<install_point>**. A command sequence similar to the following should be used:

```
cd gcc-<GCC_Version>
CC=gcc ./configure --host=<host_configuration>
--target=<target_configuration> \
    --with-gnu-as --with-gnu-ld \
    --prefix=<install_point> \
    --exec-prefix=<install_point> \
    --local-prefix=<install_point> \
    --gxx-include-dir=<install_point>
```

If the **configure** script successfully completes, then GNU **C** may be built. This step builds the gcc package in the local directory and does **NOT** install any files. The example commands specifies that *gcc* should be used as the **C** compiler, the arguments to be used by *gcc*, and the "languages" which should be built. In this example, the protoize and unprotoize tools as well as the GNU **C** and **C++** compilers are built. Protoize and unprotoize are GNU tools which aid in the conversion of K&R **C** programs to ANSI **C** and vise-versa. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g" LANGUAGES="protoize unprotoize c c++"
```

If the build process successfully completes, then the gcc package is ready to install. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g" LANGUAGES="protoize unprotoize c c++" install
```

If this successfully completes, then the gcc package has been installed at **<install_point>**. The executable files are installed with a name which includes the target configuration to insure that the cross tools have distinct names from their native counterparts. For example, *gcc* is installed as ***<target_configuration>****-gcc*.

The documentation for the gcc package is formatted using TeX. If TeX and some supporting tools are installed on the development system, then the following command may be used to produce the documentation in the "DVI" format:

```
gmake CC="gcc" CFLAGS="-O4 -g" dvi
```

2.4.5 Installing Cygnus NEWLIB

NOTES: As of the release of **RTEMS** 3.2.0, NEWLIB did not include **RTEMS** configurations. A modified copy of NEWLIB with **RTEMS** support is available on the **RTEMS** ftp site. These modifications have been submitted to the NEWLIB maintainers and are expected to be incorporated into a future release.

The following NEWLIB configurations are used by the **RTEMS** project:

Processor Family	Target Configuration
Intel i386	i386-unknown-rtems
Intel i960	i960-unknown-rtems
Hewlett Packard PA-RISC	hppa-unknown-rtems
Motorola m68k	m68k-unknown-rtems
Solaris 2.3/SPARC	unix-solaris-rtems
HPUX/PA-RISC	unix-hp-rtems

After unarchiving the NEWLIB distribution (discussed earlier), the NEWLIB package must be configured for the desired host and target. This is accomplished by changing the current directory to the top level of the NEWLIB source tree and executing the **configure** script with the appropriate arguments. This step configures the NEWLIB source for the **<target_configuration>**. The libraries and include files will be installed at **<install_point>**. The *--verbose* option to the **./configure** script is recommended because the configuration process for NEWLIB is VERY long for some targets. In particular, configuring for the Motorola m68k family is very time-consuming as the library is configured for numerous combinations of basic CPUs (m68000 and m68020) with and without floating point support. A command sequence similar to the following should be used:

```
cd newlib-<NEWLIB_Version>
CC=gcc CFLAGS="-O4 -g" ./configure --verbose \
    --target=<target_configuration> \
        --prefix=<install_point> \
```

The *--program-transform-name* option is a commonly used option for the NEWLIB configuration script. This option is given a **sed(1)** editting command which is applied to the base name for each of the tools in the cross-compilation toolset used to compile NEWLIB. For example, the option

--program-transform-name=s/\$/020/ is used to indicate that the cross tools are installed such that their names are of the form **TOOL**020. Thus the cross toolset would include the tools *gcc020*, *ar020*, and *nm020*.

If the **configure** script successfully completes, then the GNU Binary Utilities may be built. This step builds the NEWLIB package in the local directory and does **NOT** install any files. The example commands specifies that *gcc* should be used as the **C** compiler and the arguments to be used by *gcc*. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g"
```

If the build process successfully completes, then the NEWLIB package is ready to install. A command similar to the following should be used:

```
gmake CC="gcc" CFLAGS="-O4 -g" install
```

If this successfully completes, then the NEWLIB package has been installed at **<install_point>**. The executable files are installed with a name which includes the target configuration to insure that the cross tools have distinct names from their native counterparts. For example, *ar* is installed as ***<target_configuration>****-ar*.

The documentation for the NEWLIB package is formatted using TeX. If TeX and some supporting tools are installed on the development system, then the following command may be used to produce the documentation in the "DVI" format:

```
gmake CC="gcc" CFLAGS="-O4 -g" dvi
```

2.4.6 Installing FSF GNU C Library

No instructions are provided in this document for configuring or installing glibc.

2.4.7 Installing FSF GNU Debugger

No instructions are provided in this document for configuring or installing gdb.

2.4.8 Installing FSF libg++

No instructions are provided in this document for configuring or installing libg++.

2.4.9 GNU/960 Notes

The Intel GNU/960 Toolset is a conveniently packaged collection of tools and libraries based on FSF GNU, public domain, and freely available packages with some additional tools developed by Intel. The primary advantage of this toolset is that it provides a complete development environment for the i960 family which is reasonably easy to install. The primary disadvantage to this toolset is that some of the tools and libraries in the packages collection may not be the current version.

After following the installation procedure for the GNU/960 toolset, the NEWLIB installation instructions above should be followed. NEWLIB is **RTEMS** aware and provides a reentrant **C** Library.

It is not uncommon for new versions of the tools in the GNU/960 toolset to become available between updated releases of the GNU/960 toolset can incorporate them. The simplest solution the **RTEMS** developers have arrived at to address this problem is to install the updated versions of the tools in a separate directory and place this directory before that of the GNU/960 toolset in the executable search path. This makes it possible to use the "unaltered" version of the toolset if necessary. This setup has been used by the **RTEMS** developers in the past to utilize a newer version of gcc as well as to keep the NEWLIB installation separate from the GNU/960 installation.

2.5 Installing RTEMS

RTEMS is distributed as source code. It is the responsibility of the local site to build and install **RTEMS** from source code. This is not a particularly difficult procedure and gives the user two fundamental assurances:

1. *The binaries installed match the source code.*
2. *Source code is available for examination and enhancement.*

The second assurance is particularly critical when the user has the need to know the exact behavior of a particular operation. A secondary benefit of installing **RTEMS** from source code is that the compiler options used to build **RTEMS** can be -- and are -- tailored for a particular target board.

The **RTEMS** environment is designed to allow users of any shell to fully utilize the environment. However, it is necessary to configure the environment based upon the particular development language, target processor, and target board (BSP). Without a target BSP, the tests and sample applications supplied with the **RTEMS** distribution will not successfully link. For some target processors, there is a "no_bsp" target which is a stub version of a BSP which successfully compiles and may be used to link the tests. However, code cannot actually be tested without having a target board.

2.5.1 Current RTEMS Installations

If **RTEMS** is not already installed on this computer skip this section.

---

*If there have been local modifications or enhancements to **RTEMS**, then it is the responsibility of the user to preserve those changes and merge them -- if necessary -- into a new release of **RTEMS**.*

Each release of **RTEMS** is automatically placed under a parent directory which includes the release number. For example, all components of Release 3.2.0 are placed under the parent directory **rtems-3.2.0.** This should be enough to avoid a new release of **RTEMS** being accidentally installed on top of a previous one.

After reinstalling the local additions, they should be recompiled to utilize the new header files and specifications provided with the new **RTEMS** release.

2.5.1.1 Updating Current RTEMS Applications

When upgrading to a new version of the **RTEMS** executive, existing applications may require modification due to changes made within the executive. To ease this burden, RTEMS provides utilities which aid in this updating process. Please refer to the file UPGRADE_HELP found in each language specific directory under the parent directory. This file contains instructions on the use of the update utilities and other hints on updating current **RTEMS** applications.

2.5.2 Modules

Modules eases the process of manipulating the environment variables required to build **RTEMS**. If, for whatever reason, you do not wish to use Modules, then it will be necessary to manually modify shell initialization files and set the required **RTEMS** shell variables "manually." In this case, the Modules files are still the best place to look for information on what variables to set and example values.

The Modules package was utilized by the **RTEMS** developers to make the initialization process shell independent. The Modules files used by the **RTEMS** developers to configure their user environment while working on a particular target board are included in the distribution. These require only simple modifications to be "localized" for the **RTEMS** developer. The modifications to these ASCII files can be made with an editor such as vi or emacs.

2.5.2.1 New Modules Users

Install Modules on the development system following the instructions in the file **README** in the main Modules directory. When Modules has been installed and a user account setup to use Modules, then proceed to the section **Current Modules Users**.

2.5.2.2 Current Modules Users

If the host computer already utilizes the Modules package, then the assistance of the system administrator responsible for the Modules package will be required to perform the modifications described below.

The system administrator must integrate the **RTEMS** modules into the existing Modules configuration. A first alternative is to copy all of the module files provided with **RTEMS** into an existing modulefiles directory. This requires that future updates of **RTEMS** modules will also have to be integrated into the existing modulefiles directory. This alternative is not recommended.

The more preferable alternative is to add the **RTEMS** modulefiles directory to the MODULEPATH. This can be accomplished on an individual user or system wide basis. For individual users, the **RTEMS** modules directory can be added to the MODULEPATH of those users requiring access to **RTEMS**. The default MODULEPATH is established by the Modules initialization routine. Thus, the **RTEMS** modules directory must be added to the MODULEPATH in the user's shell initialization file (~/.profile or ~/.cshrc for example) following the Modules initialization statement. The proper way to accomplish this is to use the Modules **use** statement. For example, the following line should be added to the user's shell initialization file:

```
module use <rtems_path>/modules/modulefiles
```

For system wide access, the **RTEMS** modulefiles directory can be added to each of the shell initialization scripts in the existing Modules package. This will require modifying the initialization of MODULEPATH in each shell initialization file.

After integrating **RTEMS** modules with the existing modules, one may proceed to the **Configuring An RTEMS User** section.

2.5.3 Installing the C Implementation of RTEMS

After unarchiving the **RTEMS** distribution (discussed earlier), the user environment must be setup to build and install **RTEMS**. For some combinations of host development systems and targets, additional effort may be required.

2.5.3.1 Target Processor

This discussion assumes that the target processor's family is supported by this release of **RTEMS**. If the processor family is not support, **RTEMS** will have to be ported to the new architecture. This document makes no attempt to describe that process.

The list of currently supported processor families is in the **RTEMS Executive Status** section or, alternately, one may examine the source code itself. The processor family specific executive source code may be found in the subdirectories under the **c/src/exec/cpu** directory. The file **c/src/exec/cpu/<CPU>/<CPU>.h** where **<CPU>** is the appropriate processor family contains the definitions of the macros which configure the processor specific code for particular implementations within a processor family. Thus to determine if there is currently a configuration of **RTEMS** for the Motorola MC68000, one would first note that the directory **c/src/exec/cpu/m68k** does exist. Second, by examining the file **c/src/exec/cpu/m68k.h** one would note that there is already a configuration for the Motorola MC68000. [And looking farther into the source code for Release 3.2.0, one would notice that it is not completely supported.]

If the target processor family is supported but the particular processor implementation is not, then it will be necessary to add a set of entries to the file **c/src/exec/cpu/<CPU>.h** where **<CPU>** is the appropriate processor family. For each processor family, there is a set of variables which are generally boolean in nature -- they may only have 0 or 1 as their value. Each variable corresponds to a feature whose presence varies amongst implementations of the processor family. The most common feature to be selectively enabled is support for floating point hardware such as a coprocessor.

The use of feature based conditional compilation makes it surprisingly easy to add support for a new implementation within a processor family. The traditional approach to this problem is for each individual conditional compilation statement to have a long, complicated list of processors which support the feature. This often makes it difficult to determine what exactly is being enabled or disabled and, even worse, the addition of support for a new processor implementation requires the examination of each conditional compilation statement individually. The **RTEMS** approach reduces the problem of adding support for a new processor implementation to simply identifying whether a particular processor family feature is present or absent.

2.5.3.2 Host Operating System

The **RTEMS** Development Environment is carefully designed to be configurable to operate on as many operating systems as possible. The primary requirement is that the GNU toolset be available on that host. It is the responsibility of the local installation to install the required native and cross development tools.

If you are on a UNIX™ host, then the easiest way on most systems to determine the name and revision level of the operating system is to execute the command *uname -a*. The output of this command contains the necessary information.

The list of currently supported hosts operating systems for use as a development system is reflected by the make configuration files in the directory **c/make/os**. The files contained in that directory follow a straightforward naming scheme which reflects the operating system name and revision level.

If there is not a configuration file for your host operating system, then it will be necessary to create a new one. This is not a difficult process and can usually be accomplished quite quickly if the required tools were shipped with the operating system. The host operating system configuration file sets a number of make variables to the full path of the utilities required by the **RTEMS** build process. The *type* command in derivatives of the Bourne and Korn shell and the *whereis* command in CSH derivatives may be used to determine the full path of any command.

If some tools listed in the host operating system configuration file are not available on your development system or the tool is present but old or buggy, then the best alternative is to acquire and install the FSF GNU replacement for this tool.

2.5.3.3 Target Board

The list of currently supported target boards is in the **RTEMS Executive Status** section. Alternately, one may examine the source code itself. The source code for the Board Support Packages (BSP) for a particular processor family can be found in the directory **c/src/lib/libbsp/<CPU>** where **<CPU>** is the name of the processor family. Note that many targets have multiple part names and that it may be necessary to look for both the desired target and its "cousins." For example, the DY-4 SVME153 is the commercial version of the DY-4 DVME152 VMEbus Motorola MC68020 board. There is a BSP directory for the DMV152 but not the SVME153 board.

If the target board is not supported, then it will be necessary to perform the following tasks:

- *develop a Board Support Package (BSP)*
- *develop the following Makefile configuration files:*
  - compiler configuration file
  - general customization file
- *develop a Modules file*

The organization of a Modules file is discussed in the next section.

2.5.3.3.1 Board Support Package

A Board Support Package (BSP) is a collection of device drivers, initialization code, and linker scripts necessary to execute **RTEMS** on a particular target board. The minimum set of device drivers for a single processor target includes a Clock, Console I/O, and Benchmark Timer device drivers. A multiprocessor target adds some Multiprocessor Communications Interface (MPCI) -- typically the small target specific support for the **RTEMS** Shared Memory Device Driver. The source code for a BSP is found in the directory **c/src/lib/libbsp/<CPU>/<TARGET>** where **<CPU>** is the processor family and **<TARGET>** is the name of the target board. For example, the Motorola MC68020-based Motorola MVME136 BSP is in the directory **c/src/lib/libbsp/m68k/mvme136**.

2.5.3.3.2 Makefile Configuration Files

There are two target specific configuration files used by the Makefile system. These configuration files specify detailed information about the toolset, the compilation process, as well as some general configuration information regarding the target and the development environment. The following is a list of these configuration files:

- *c/make/compilers/gcc-**<TARGET>**.cfg*
- *c/make/custom/**<TARGET>**.cfg*

The recommended procedure is to make a copy of the configuration files for a target which is reasonably close to the one you will be using and edit them as appropriate. If the processor implementations are the same on the two target boards, then the changes should be very minor.

2.5.3.3.3 Creating a Customized Modules File

Files which the Modules packages may use to customize a user's environment for building, installing, and modifying **RTEMS** are found in the **c/Modules/rtems** directory. Each of the files in this directory corresponds to the configuration used by the **RTEMS** developers for building and installing **RTEMS** for a particular target board. These files contain the Modules commands necessary to set the following environment variables:

Variable	Description
RTEMS_BSP	The name of the target BSP (e.g. mvme136 or cvme961).
RTEMS_ROOT	The full path of root directory of the **RTEMS** source code.
RTEMS_GNUTOOLS	The full path of the root directory for the cross-development toolset to be used.
RTEMS_HOST	The name of the operating system for the development system.
RTEMS_LIBC_DIR	The full path of the root directory for the Standard C Library to be used.

2.5.3.4 Configuring an RTEMS User

Each user building and installing **RTEMS** must have their environment configured. The user environment must have a set of variables set in it which indicate the target BSP, host operating system, and numerous paths. This section discusses the process of setting up a user environment to build and install **RTEMS**.

2.5.3.4.1 Configuring an RTEMS User Using Modules

If a user will always use a particular configuration (language, processor, and target board) then a line of the following type may be added to the initialization file for their shell after the Modules initialization statement.

```
module load oar-mvme136
```

where *oar-mvme136* is the name of the Module filed used by the **RTEMS** developers to initialize the environment for using the Motorola MVME136 BSP. The user must logout and login before any changes to the shell initialization files will take effect. [NOTE: The Modules files included with the **RTEMS** distribution **MUST** be editted before being used on the user's development system. These files contain information regarding the host OS of the development machine and the full path of the root of the **RTEMS** source code.]

If the user does not wish the **RTEMS** environment configuration to be added to their shell initialization file, then the "module load" statement may be entered at the command line.

The following examples illustrate the "module load" statement. The first uses Modules to configure the user's environment to use a copy of the standard MVME136 Modules files while has been tailored to reflect the local installation of **RTEMS**:

```
module load local-mvme136
```

Although each user may specify a different **RTEMS** configuration, only one **RTEMS** configuration may be active at any particular time for a given user. In addition, multiple users with differing configurations loaded may use the same **RTEMS** source tree simultaneously. This capability is used by the **RTEMS** developers to simultaneously compile **RTEMS** for multiple processors and targets simultaneously.

A user may switch from one **RTEMS** configuration to another with either of the following command sequences:

```
module unload <old_rtems_modulefile>
module load <new_rtems_modulefile>
                              OR
module switch <old_rtems_modulefile> <new_rtems_modulefile>
```

The command "module avail" may be used to obtain a list of the Module files which are available to be loaded using the "module load" command.

The command "module list" provides a list of the currently loaded Modules.

2.5.3.4.2 Configuring a RTEMS User NOT Using Modules

For **RTEMS** users who are not Modules users, the shell variables set in the Modules file must be set. It is recommended that non-Modules users create a Modules file anyway and simply mimic the changes which would be made to the user environment by that Modules file. The exact operations required to accomplish these changes are shell dependent.

2.5.3.5 Building RTEMS

By default, **RTEMS** is installed into the directory **c/<TARGET>**. If this is not the desired installation point, then modify the following line in the file **c/make/custom/<TARGET>.cfg**:

```
PROJECT_HOME=$(PROJECT_ROOT)/$(RTEMS_BSP)
```

Once the root directory of the install point has been set, the following steps are required to build and install **RTEMS** [NOTE: If the Modules files are used, the environment variable **$r** is set to point to the top directory for the **RTEMS**

implementation selected in the current environment. This is the directory **c** in the **RTEMS** distribution.] :

```
cd $r
gmake install
```

If this completes successfully, then **RTEMS** has been built and installed.

2.5.4 Installing the Ada Implementation of RTEMS

After unarchiving the **RTEMS** distribution (discussed earlier), the user environment must be setup to build and install **RTEMS**. For some combinations of host development systems and targets, additional effort may be required.

[NOTE: Currently the **Ada** implementation of **RTEMS** only supports the Tartan MC68xxx **Ada** cross-compiler. Please refer to the Tartan documentation for instructions on installing that toolset. In addition, The **RTEMS** build procedure using this compiler is not as simple as the corresponding procedure for the **C** implementation. Initial tests indicate that once the transition has been made to GNU **Ada** (GNAT), the build procedure for the **Ada** implementation will be as simple as that for the **C** implementation.]

2.5.4.1 Target Processor

This discussion assumes that the target processor's family is supported by this release of **RTEMS**. If the processor family is not support, **RTEMS** will have to be ported to the new architecture. This document makes no attempt to describe that process.

The list of currently supported processor families and compilers is in the **RTEMS Executive Status** section or, alternately, one may examine the source code itself. The processor family specific executive source code may be found in the subdirectories under the **ada/src/exec/<COMPILER>/cpu** directory with **<COMPILER>** indicating the **Ada** compiler used by the port. The **ada/make/<COMPILER>/cpu** directory contains the files included from the makefiles which configure the processor specific code for particular implementations within a processor family. Thus to determine if there is currently a configuration of **RTEMS** for the Motorola MC68000 using the GNU **Ada** compiler, one would first note that the directory **ada/src/exec/cpu/gnu/m68k** does exist. Second, by examining the file **ada/make/gnu/cpu/m68k.cfg** one would note that there is already a configuration for the Motorola MC68000.

[NOTE: The current release of the **Ada** implementation of **RTEMS** does not support the concept of a processor implementation within a family as discussed in the these paragraphs. The lack of conditional compilation makes it difficult to carry this concept efficiently from the **C** to **Ada** implementation.]

If the target processor family is supported but the particular processor implementation is not, then it will be necessary to add a set of entries to the file **ada/make/<COMPILER>/cpu/<CPU>.cfg** where **<COMPILER>** is the appropriate compiler and **<CPU>** is the appropriate processor family. For each processor family, there is a set of variables which are generally boolean in nature -- they may only have 0 or 1 as their value. Each variable corresponds to a feature whose presence varies amongst implementations of the processor family. The most common feature to be selectively enabled is support for floating point hardware such as a coprocessor.

The use of feature based conditional compilation makes it surprisingly easy to add support for a new implementation within a processor family. The traditional approach to this problem is for each individual conditional compilation statement to have a long complicated list of processors which support the feature. This often makes it difficult to determine what exactly is being enabled or disabled and, even worse, the addition of support for a new processor implementation requires the examination of each conditional compilation statement individually. The **RTEMS** approach reduces the problem of adding support for a new processor implementation to simply identifying whether a particular processor family feature is present or absent.

2.5.4.2 Host Operating System

The **RTEMS** Development Environment is carefully designed to be configurable to operate on as many operating systems as possible. The primary requirement is that the GNU toolset be available on that host.

If you are on a UNIX™ host, then the easiest way on most systems to determine the name and revision level of the operating system is to execute the command *uname -a*. The output of this command contains the necessary information.

The list of currently supported hosts operating systems for use as a development system is reflected by the make configuration files in the directory **ada/make/<COMPILER>/os** where **<COMPILER>** is the compiler in use. The files contained in that directory follow a straightforward naming scheme which reflects the operating system name and revision level.

If there is not a configuration file for your host operating system, then it will be necessary to create a new one. This is not a difficult process and can usually be accomplished quite quickly if the required tools were shipped with the operating system. The host operating system configuration file sets a number of make variables to the full path of the utilities required by the **RTEMS** build process. The *type* command in derivatives of the Bourne and Korn shell and the

whereis command in CSH derivatives may be used to determine the full path of any command.

If some tools listed in the host operating system configuration file are not available on your development system or the tool is present but old or buggy, then the best alternative is to acquire and install the FSF GNU replacement for this tool.

2.5.4.3 Target Board

The list of currently supported target boards is in the **RTEMS Executive Status** section. Alternately, one may examine the source code itself. The source code for the Board Support Packages (BSP) for a particular processor family can be found in the directory **ada/src/lib/libbsp/<COMPILER>/<CPU>** where **<COMPILER>** is the compiler in use and **<CPU>** is the name of the processor family. Note that many targets have multiple part names and that it may be necessary to look for both the desired target and its "cousins." For example, the DY-4 SVME153 is the commercial version of the DY-4 DVME152 VMEbus Motorola MC68020 board. There is a BSP directory for the DMV152 but not the SVME153 board.

If the target board is not supported, then it will be necessary to perform the following tasks:

- *develop a Board Support Package (BSP)*
- *develop the following Makefile configuration files:*
  - compiler configuration file
  - general customization file
- *develop a Modules file*

The organization of a Modules file is discussed in the next section.

2.5.4.3.1 Board Support Package

A Board Support Package (BSP) is a collection of device drivers, initialization code, and linker scripts necessary to execute **RTEMS** on a particular target board. The minimum set of device drivers for a single processor target includes a Clock, Console I/O, and Benchmark Timer device drivers. A multiprocessor target adds some Multiprocessor Communications Interface (MPCI) -- typically the small target specific support for the **RTEMS** Shared Memory Device Driver. The source code for a BSP is found in the directory **ada/src/lib/libbsp/<COMPILER>/<CPU>/<TARGET>** where **<COMPILER>** is the compiler in use, **<CPU>** is the processor family, and **<TARGET>** is the

name of the target board. For example, the Motorola MC68020-based Motorola MVME136 BSP for GNU **Ada** is in the directory **ada/src/lib/libbsp/gnu/m68k/mvme136**.

2.5.4.3.2 Makefile Configuration Files

There are two target specific configuration files used by the Makefile system. These configuration files specify detailed information about the toolset, the compilation process, as well as some general configuration information regarding the target and the development environment. The following is a list of these configuration files:

- *ada/make/**<COMPILER>**/compilers/gcc-**<TARGET>**.cfg*
- *ada/make/**<COMPILER>**/custom/**<TARGET>**.cfg*

The recommended procedure is to make a copy of the configuration files for a target which is reasonably close to the one you will be using and edit them as appropriate. If the processor implementations are the same on the two target boards, then the changes should be very minor.

[NOTE: The make configuration files for the Tartan **Ada** toolset do not support the compilers configuration file.]

2.5.4.3.3 Creating a Customized Modules File

Files which the Modules packages may use to customize a user's environment for building, installing, and modifying **RTEMS** are found in the **ada/Modules/rtems** directory. Each of the files in this directory corresponds to the configuration used by the **RTEMS** developers for building and installing **RTEMS** for a particular target board. These files contain the Modules commands necessary to set the following environment variables:

2.5.4.4 Configuring an RTEMS User

Each user building and installing **RTEMS** must have their environment configured. The user environment must have a set of variables set in it which indicate the target BSP, host operating system, and numerous paths. This section discusses the process of setting up a user environment to build and install **RTEMS**.

2.5.4.4.1 Configuring an RTEMS User Using Modules

If a user will always use a particular configuration (language, processor, and target board) then a line of the following type may be added to the initialization file for their shell after the Modules initialization statement.

```
module load oar-mvme136
```

where *oar-mvme136* is the name of the Module filed used by the **RTEMS** developers to initialize the environment for using the Motorola MVME136 BSP. The user must logout and login before any changes to the shell initialization files will take effect. [NOTE: The Modules files included with the **RTEMS** distribution **MUST** be editted before being used on the user's development system. These files contain information regarding the host OS of the development machine and the full path of the root of the **RTEMS** source code.]

If the user does not wish the **RTEMS** environment configuration to be added to their shell initialization file, then the "module load" statement may be entered at the command line.

The following examples illustrate the "module load" statement. The first uses Modules to configure the user's environment to use a copy of the standard MVME136 Modules files while has been tailored to reflect the local installation of **RTEMS**:

```
module load local-mvme136
```

Although each user may specify a different **RTEMS** configuration, only one **RTEMS** configuration may be active at any particular time for a given user. In addition, multiple users with differing configurations loaded may use the same **RTEMS** source tree simultaneously. This capability is used by the **RTEMS** developers to simultaneously compile **RTEMS** for multiple processors and targets simultaneously.

A user may switch from one **RTEMS** configuration to another with either of the following command sequences:

```
module unload <old_rtems_modulefile>
module load <new_rtems_modulefile>
                              OR
module switch <old_rtems_modulefile> <new_rtems_modulefile>
```

The command "module avail" may be used to obtain a list of the Module files which are available to be loaded using the "module load" command.

The command "module list" provides a list of the currently loaded Modules.

2.5.4.4.2 Configuring a RTEMS User NOT Using Modules

For **RTEMS** users who are not Modules users, the shell variables set in the Modules file must be set. It is recommended that non-Modules users create a Modules file anyway and simply mimic the changes which would be made to the user environment by that Modules file. The exact operations required to accomplish these changes are shell dependent.

2.5.4.5 Building RTEMS

[NOTE: This procedure does not apply to the Tartan compiler. Please refer to the section **Building RTEMS Using the Tartan Compiler**.]

By default, **RTEMS** is installed into the directory **ada/<TARGET>**. If this is not the desired installation point, then modify the following line in the file **ada/make/<COMPILER>/custom/<TARGET>.cfg**:

```
PROJECT_HOME=$(PROJECT_ROOT)/$(RTEMS_BSP)
```

Once the root directory of the install point has been set, the following steps are required to build and install **RTEMS** [NOTE: If the Modules files are used, the environment variable **$r** is set to point to the top directory for the **RTEMS** implementation selected in the current environment. This is the directory **c** in the **RTEMS** distribution.] :

```
cd $r
gmake install
```

If this completes successfully, then **RTEMS** has been built and installed.

2.5.4.6 Building RTEMS Using the Tartan Compiler

This section describes the process of building **RTEMS** using the Tartan **Ada** compiler once the desired BSP is in place and the user environment is initialized. [NOTE: This procedure has only been tested on version 4.3.1 of the Tartan MC68xxx **Ada** Cross-Compiler hosted on a Sun SparcStation running the Solaris 2.3 operating system.]

The file **ada/make/tartan/baserule** contains most of the Makefile variables which might be changed for the local installation. In particular, this file sets the project name. In addition, the user should examine the rules for assembling and importing Assembly source code into the Tartan libraries. Finally, the files in the directory **ada/src/exec/cpu/tartan/tools** should be examined and localized. The primary tool of interest in this directory is *scope*, a utility which eases the burden of invoking Tartan AdaScope for each individual test.

Once the Makefile subsystem and tools have been tailored, the following steps may be taken to build the Motorola MC68020 port of **RTEMS** for the Tartan **Ada** compiler. [NOTE: If the Modules files are used, the environment variable **$r** is set to point to the top directory for the **RTEMS** implementation selected in the current environment. This is the directory **c** in the **RTEMS** distribution.] :

1. *cd $r/src/exec/rtems*
2. *gmake mkprj*
3. *cd $r/src/exec/rtems*
4. *gmake mklib import all freeze*
5. *cd $r/src/lib*
6. *gmake mklib add*
7. *cd $r/src/tests*
8. *gmake mklib import add all*

If this completes successfully, then **RTEMS** has been built and installed.