# Getting Started with GNAT/RTEMS

Edition 4.8.2, for 4.8.2

18 March 2010

**On-Line Applications Research Corporation**

On-Line Applications Research Corporation
TEXinfo 2006-10-04.17

COPYRIGHT © 1988 - 2007.
On-Line Applications Research Corporation (OAR).

The authors have used their best efforts in preparing this material. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. No warranty of any kind, expressed or implied, with regard to the software or the material contained in this document is provided. No liability arising out of the application or use of any product described in this document is assumed. The authors reserve the right to revise this material and to make changes from time to time in the content hereof without obligation to notify anyone of such revision or changes.

The RTEMS Project is hosted at `http://www.rtems.com`. Any inquiries concerning RTEMS, its related support components, its documentation, or any custom services for RTEMS should be directed to the contacts listed on that site. A current list of RTEMS Support Providers is at `http://www.rtems.com/support.html`.

# Table of Contents


**1 Introduction ..... 1**

1.1 Real-Time Embedded Systems ..... 1
1.2 Cross Development ..... 2
1.3 Resources on the Internet ..... 3
1.3.1 RTEMS Mailing List ..... 3
1.3.2 CrossGCC Mailing List ..... 3
1.3.3 GNAT Chat Mailing List ..... 3

**2 Requirements ..... 5**

2.1 Native GNAT ..... 5
2.1.1 Verifying Correct Operation of Native GNAT ..... 6
2.1.1.1 Native Hello World Test ..... 6
2.1.1.2 Insure GCC and GNAT Environment Variables Are Not Set ..... 6

**3 Building the GNAT Cross Compiler Toolset ..... 7**

3.1 Create the Archive and Build Directories ..... 7
3.2 Get All the Pieces ..... 7
3.3 Unarchiving the Tools ..... 8
3.4 Host Specific Notes ..... 9
3.4.1 Solaris 2.x ..... 9
3.5 Reading the Tools Documentation ..... 9
3.6 Apply RTEMS Patch to GCC ..... 10
3.7 Apply RTEMS Patch to binutils ..... 10
3.8 Apply RTEMS Patch to newlib ..... 10
3.9 Apply RTEMS Patch to GNAT ..... 10
3.10 Copy the ada Subdirectory to the GCC Source Tree ..... 11
3.11 Localizing the Configuration ..... 11
3.12 Running the bit_ada Script ..... 13
3.13 Common Problems ..... 14
3.13.1 Error Message Indicates Invalid Option to Assembler ..... 14
3.13.2 Error Messages Indicating Configuration Problems ..... 14

**4 Building RTEMS ..... 17**

4.1 Obtain the RTEMS Source Code ..... 17
4.2 Unarchive the RTEMS Source ..... 17
4.3 Add <INSTALL_POINT>/bin to Executable PATH ..... 17
4.4 Verifying the Operation of the Cross Toolset ..... 17
4.5 Building RTEMS for a Specific Target and BSP ..... 18
4.5.1 Using the RTEMS configure Script Directly ..... 18

**5 Building the Sample Application ..... 21**
5.1 Unpack the Sample Application ..... 21
5.2 Create a BSP Specific Makefile ..... 21
5.3 Build the Sample Application ..... 21
5.4 Application Executable ..... 21

**6 Building the GNU Debugger ..... 23**
6.1 Unarchive the gdb Distribution ..... 23
6.2 Apply GNAT Patch to GDB ..... 23
6.3 Apply RTEMS Patch to GDB ..... 23
6.4 GDB with Sparc Instruction Simulation (SIS) ..... 23
6.5 GDB with PowerPC Instruction Simulator ..... 24
6.6 GDB for DINK32 ..... 25

# 1 Introduction

The purpose of this document is to guide you through the process of installing a GNU cross development environment to use with RTEMS.

If you are already familiar with the concepts behind a cross compiler and have a background in Unix, these instructions should provide the bare essentials for performing a setup of the following items:

- GNAT/RTEMS Cross Compilation Tools on your host system
- RTEMS OS for the target host
- GDB Debugger

The remainder of this chapter provides background information on real-time embedded systems and cross development and an overview of other resources of interest on the Internet. If you are not familiar with real-time embedded systems or the other areas, please read those sections. These sections will help familiarize you with the types of systems RTEMS is designed to be used in and the cross development process used when developing RTEMS applications.

## 1.1 Real-Time Embedded Systems

Real-time embedded systems are found in practically every facet of our everyday lives. Today's systems range from the common telephone, automobile control systems, and kitchen appliances to complex air traffic control systems, military weapon systems, an d production line control including robotics and automation. However, in the current climate of rapidly changing technology, it is difficult to reach a consensus on the definition of a real-time embedded system. Hardware costs are continuing to rapidly decline while at the same time the hardware is increasing in power and functionality. As a result, embedded systems that were not considered viable two years ago are suddenly a cost effective solution. In this domain, it is not uncommon for a single hardware configuration to employ a variety of architectures and technologies. Therefore, we shall define an embedded system as any computer system that is built into a larger system consisting of multiple technologies such as digital and analog electronics, mechanical devices, and sensors.

Even as hardware platforms become more powerful, most embedded systems are critically dependent on the real-time software embedded in the systems themselves. Regardless of how efficiently the hardware operates, the performance of the embedded real-time software determines the success of the system. As the complexity of the embedded hardware platform grows, so does the size and complexity of the embedded software. Software systems must routinely perform activities which were only dreamed of a short time ago. These large, complex, real-time embedded applications now commonly contain one million lines of code or more.

Real-time embedded systems have a complex set of characteristics that distinguish them from other software applications. Real-time embedded systems are driven by and must respond to real world events while adhering to rigorous requirements imposed by the environment with which they interact. The correctness of the system depends not only on the results of computations, but also on the time at which the results are produced. The most

important and complex characteristic of real-time application systems is that they must receive and respond to a set of external stimuli within rigid and critical time constraints.

A single real-time application can be composed of both soft and hard real-time components. A typical example of a hard real-time system is a nuclear reactor control system that must not only detect failures, but must also respond quickly enough to prevent a meltdown. This application also has soft real-time requirements because it may involve a man-machine interface. Providing an interactive input to the control system is not as critical as setting off an alarm to indicate a failure condition. However, th e interactive system component must respond within an acceptable time limit to allow the operator to interact efficiently with the control system.

## 1.2 Cross Development

Today almost all real-time embedded software systems are developed in a **cross development** environment using cross development tools. In the cross development environment, software development activities are typically performed on one computer system, the **host** system, while the result of the development effort (produced by the cross tools) is a software system that executes on the **target** platform. The requirements for the target platform are usually incompatible and quite often in direct conflict with the requirements for the host. Moreover, the target hardware is often custom designed for a particular project. This means that the cross development toolset must allow the developer to customize the tools to address target specific run-time issues. The toolset must have provisions for board dependent initialization code, device drivers, and error handling code.

The host computer is optimized to support the code development cycle with support for code editors, compilers, and linkers requiring large disk drives, user development windows, and multiple developer connections. Thus the host computer is typically a traditional UNIX workstation such as are available from SUN or Silicon Graphics, or a PC running either a version of MS-Windows or UNIX. The host system may also be required to execute office productivity applications to allow the software developer to write documentation, make presentations, or track the project's progress using a project management tool. This necessitates that the host computer be general purpose with resources such as a thirty-two or sixty-four bit processor, large amounts of RAM, a monitor, mouse, keyboard, hard and floppy disk drives, CD-ROM drive, and a graphics card. It is likely that the system will be multimedia capable and have some networking capability.

Conversely, the target platform generally has limited traditional computer resources. The hardware is designed for the particular functionality and requirements of the embedded system and optimized to perform those tasks effectively. Instead of hard driversΦΦs and keyboards, it is composed of sensors, relays, and stepper motors. The per-unit cost of the target platform is typically a critical concern. No hardware component is included without being cost justified. As a result, the processor of the target system is often from a different processor family than that of the host system and usually has lower performance. In addition to the processor families targeted only for use in embedded systems, there are versions of nearly every general-purpose process or specifically tailored for real-time embedded systems. For example, many of the processors targeting the embedded market do not include hardware floating point units, but do include peripherals such as timers, serial controllers, or network interfaces.

## 1.3 Resources on the Internet

This section describes various resources on the Internet which are of use to GNAT/RTEMS users.

### 1.3.1 RTEMS Mailing List

rtems-users@rtems.com

This mailing list is dedicated to the discussion of issues related to RTEMS, including GNAT/RTEMS. If you have questions about RTEMS, wish to make suggestions, or just want to pick up hints, this is a good list to subscribe to. Subscribe by sending an empty mail message to rtems-users-subscribe@rtems.com. Messages sent to rtems-users@rtems.com are posted to the list.

### 1.3.2 CrossGCC Mailing List

crossgcc@cygnus.com

This mailing list is dedicated to the use of the GNU tools in cross development environments. Most of the discussions focus on embedded issues. Subscribe by sending a message with the one line "subscribe" to crossgcc-request@cygnus.com.

The crossgcc FAQ as well as a number of patches and utiliities of interest to cross development system users are available at ftp://ftp.cygnus.com/pub/embedded/crossgcc.

### 1.3.3 GNAT Chat Mailing List

chat@gnat.com

This mailing list is dedicated to the general discussion of GNAT specific issues. The discussions try to avoid more general Ada95 language issues which have other forums. Subscribe by sending a message with the one line "subscribe" to chat-request@gnat.com.

# 2 Requirements

A fairly large amount of disk space is required to perform the build of the GNU C/C++ Cross Compiler Tools for RTEMS. The following table may help in assessing the amount of disk space required for your installation:

| Component | Disk Space Required |
|---|---|
| archive directory | 40 Mbytes |
| tools src unarchived | 200 Mbytes |
| each individual build directory | up to 500 Mbytes |
| each installation directory | 20-200 Mbytes |

It is important to understand that the above requirements only address the GNU C/C++ Cross Compiler Tools themselves. Adding additional languages such as Fortran or Objective-C can increase the size of the build and installation directories. Also, the unarchived source and build directories can be removed after the tools are installed.

After the tools themselves are installed, RTEMS must be built and installed for each Board Support Package that you wish to use. Thus the precise amount of disk space required for each installation directory depends highly on the number of RTEMS BSPs which are to be installed. If a single BSP is installed, then the additional size of each install directory will tend to be in the 40-60 Mbyte range.

There are a number of factors which must be taken into account in oreder to estimate the amount of disk space required to build RTEMS itself. Attempting to build multiple BSPs in a single step increases the disk space requirements. Similarly enabling optional features increases the build and install space requirements. In particular, enabling and building the RTEMS tests results in a significant increase in build space requirements but since the test are not installed has no impact on installation requirements.

The instructions in this manual should work on any computer running a UNIX variant. Some native GNU tools are used by this procedure including:

- GCC
- GNAT
- GNU make

In addition, some native utilities may be deficient for building the GNU tools.

## 2.1 Native GNAT

The native GNAT must be installed in the default location or built from source. No GCC or GNAT environment variables should be set during the build or use of the cross GNAT/RTEMS toolset as this could result in an unpredictable mix of native and cross toolsets.

Binaries for native GNAT installations are available at the primary GNAT ftp site ([No value for “GNAT-FTP”]. Installation instructions are included with the binary GNAT dis-

tributions. The binary installation should be installed in the default location or installed in a non-default location and used ONLY to build a native GNAT from source. This final native GNAT will be used to build the GNAT/RTEMS cross development toolset.

### 2.1.1 Verifying Correct Operation of Native GNAT

It is imperative that the native GNAT installation work correctly for the installation of GNAT/RTEMS to succeed. It is recommended that the user verify that the native GNAT is installed correctly by performing these tests:

#### 2.1.1.1 Native Hello World Test

Place the following Ada source code in hello.adb:

```
with Text_IO; use Text_IO;

procedure Hello is
begin
   Put_Line ( "Hello World");
end Hello;
```

Use the following command sequence to ompile and execute the above program:

```
gnatmake hello
./hello
```

If the message `Hello World` is printed, then the native installation of GNAT operates well enough to proceed.

#### 2.1.1.2 Insure GCC and GNAT Environment Variables Are Not Set

If any of the following commands produce output, then you have environment variables overriding the default behavior of the native GNAT toolset. These variables will conflict with the cross toolset. Please resolve this problem before proceeding further.

```
echo $GCC_EXEC_PREFIX
echo $ADA_INCLUDE_PATH
echo $ADA_OBJECTS_PATH
echo $LD_RUN_PATH
echo $C_INCLUDE_PATH
```

# 3 Building the GNAT Cross Compiler Toolset

This chapter describes the steps required to acquire the source code for a GNU cross compiler toolset, apply any required RTEMS specific patches, compile that toolset and install it.

## 3.1 Create the Archive and Build Directories

Start by making the `archive` directory to contain the downloaded source code and the `tools` directory to be used as a build directory. The command sequence to do this is shown below:

```
mkdir archive
mkdir tools
```

This will result in an initial directory structure similar to the one shown in the following figure:

```
/whatever/prefix/you/choose/
        archive/
        tools/
```

## 3.2 Get All the Pieces

This section lists the components of an RTEMS cross development system. Included are the locations of each component as well as any required RTEMS specific patches.

### gcc 2.8.1

```
FTP Site:    ftp.gnu.org
Directory:   /pub/gnu/gcc
File:        gcc-2.8.1.tar.gz
```

### gnat 3.13p

```
FTP Site:    NONE
Directory:   NO_DIRECTORY
File:        gnat-3.13p-src.tar.gz
```

### binutils 2.10

```
FTP Site:    ftp.gnu.org
Directory:   /pub/gnu/binutils
File:        binutils-2.10.tar.gz
```

### newlib 1.8.2

```
FTP Site:    sources.redhat.com
Directory:   /pub/newlib
File:        newlib-1.8.2.tar.gz
```

### RTEMS Snapshot

```
FTP Site:    ftp.rtems.com
Directory:   /pub/rtems/snapshots/current
File:        rtems-ss-DATE.tgz
```

### RTEMS Hello World

```
FTP Site:    ftp.rtems.com
Directory:   /pub/rtems/snapshots/current
File:        hello_world_ada.tgz
```

### RTEMS Specific Tool Patches and Scripts

```
FTP Site:    ftp.rtems.com
Directory:   /pub/rtems/snapshots/current/ada_tools/source
File:        [No value for ``BUILDTOOLSTAR'']
File:        binutils-2.10-rtems-gnat-3.13p-20001107.diff
File:        newlib-1.8.2-rtems-20000606.diff.gz
File:        gcc-2.8.1-rtems-gnat-3.13p-20000429.diff.gz
File:        gnat-3.13p-rtems-20000829.diff
```

## 3.3 Unarchiving the Tools

While in the `tools` directory, unpack the compressed tar files using the following command sequence:

```
cd tools
tar xzf ../archive/gcc-2.8.1.tar.gz
tar xzf ../archive/gnat-3.13p-src.tar.gz
tar xzf ../archive/binutils-2.10.tar.gz
tar xzf ../archive/newlib-1.8.2.tar.gz
tar xzf ../archive/[No value for ``BUILDTOOLSTAR'']
```

After the compressed tar files have been unpacked, the following directories will have been created under tools.

- binutils-2.10
- gcc-2.8.1
- gnat-3.13p-src
- newlib-1.8.2

There will also be a set of scripts in the current directory which aid in building the tools and RTEMS. They are:

- bit_ada
- bit_gdb
- bit_rtems
- common.sh
- user.cfg

When the `bit_ada` script is executed later in this process, it will automatically create two other subdirectories:

- src
- build-${CPU}-tools

Similarly, the `bit_gdb` script will create the subdirectory `build-${CPU}-gdb` and the `bit_rtems` script will create the subdirectory `build-${CPU}-rtems`.

The directory tree should look something like the following figure:

```
/whatever/prefix/you/choose/
        archive/
            gcc-2.8.1.tar.gz
            gnat-3.13p-src.tar.gz
            binutils-2.10.tar.gz
            newlib-1.8.2.tar.gz
            rtems-ss-DATE.tgz
            [No value for ``BUILDTOOLSTAR'']
            gcc-2.8.1-rtems-gnat-3.13p-20000429.diff.gz
            binutils-2.10-rtems-gnat-3.13p-20001107.diff
            newlib-1.8.2-rtems-20000606.diff.gz
            gnat-3.13p-rtems-20000829.diff
            hello_world_ada.tgz
            bit_ada
        tools/
            binutils-2.10/
            gcc-2.8.1/
            gnat-3.13p-src/
            newlib-1.8.2/
            bit_ada
            bit_gdb
            bit_rtems
            common.sh
            user.cfg
```

## 3.4 Host Specific Notes

### 3.4.1 Solaris 2.x

The build scripts are written in "shell". The program `/bin/sh` on Solaris 2.x is not robust enough to execute these scripts. If you are on a Solaris 2.x host, then change the first line of the files `bit_ada`, `bit_gdb`, and `bit_rtems` to use the `/bin/ksh` shell instead.

## 3.5 Reading the Tools Documentation

Each of the tools in the GNU development suite comes with documentation. It is in the reader's and tool maintainers' interest that one read the documentation before posting a problem to a mailing list or news group.

## 3.6 Apply RTEMS Patch to GCC

Apply the patch using the following command sequence:

```
cd tools/gcc-2.8.1
zcat ../../archive/gcc-2.8.1-rtems-gnat-3.13p-20000429.diff.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/gcc-2.8.1
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

## 3.7 Apply RTEMS Patch to binutils

Apply the patch using the following command sequence:

```
cd tools/binutils-2.10
zcat ../../archive/binutils-2.10-rtems-gnat-3.13p-20001107.diff | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/binutils-2.10
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

## 3.8 Apply RTEMS Patch to newlib

Apply the patch using the following command sequence:

```
cd tools/newlib-1.8.2
zcat ../../archive/newlib-1.8.2-rtems-20000606.diff.gz | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/newlib-1.8.2
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

## 3.9 Apply RTEMS Patch to GNAT

Apply the patch using the following command sequence:

```
cd tools/gnat-3.13p-src
zcat ../../archive/gnat-3.13p-rtems-20000829.diff | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/gnat-3.13p-src
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file which is properly applied.

## 3.10 Copy the ada Subdirectory to the GCC Source Tree

Copy the ada subtree in the patched subtree of tools/gnat-3.13p-src/src to the tools/gcc-2.8.1 directory:

```
cd tools/gnat-3.13p-src/src
cp -r ada ../../gcc-2.8.1
```

## 3.11 Localizing the Configuration

Edit the `user.cfg` file to alter the settings of various variables which are used to tailor the build process. Each of the variables set in `user.cfg` may be modified as described below:

`INSTALL_POINT` is the location where you wish the GNU C/C++ cross compilation tools for RTEMS to be built. It is recommended that the directory chosen to receive these tools be named so that it is clear from which gcc distribution it was generated and for which target system the tools are to produce code for.

**WARNING**: The `INSTALL_POINT` should not be a subdirectory under the build directory. The build directory will be removed automatically upon successful completion of the build procedure.

`BINUTILS` is the directory under tools that contains binutils-2.10. For example:

```
BINUTILS=binutils-2.10
```

`GCC` is the directory under tools that contains gcc-2.8.1. For example,

```
GCC=gcc-2.8.1
```

Note that the gnat version is not needed because the gnat source is built as part of building gcc.

`NEWLIB` is the directory under tools that contains newlib-1.8.2. For example:

```
NEWLIB=newlib-1.8.2
```

`BUILD_DOCS` is set to "yes" if you want to install documentation. This requires that tools supporting documentation production be installed. This currently is limited to the GNU texinfo package. For example:

```
BUILD_DOCS=yes
```

`BUILD_OTHER_LANGUAGES`
is set to "yes" if you want to build languages other than C and C++. At the current time, the set of alternative languages includes Java, Fortran, and Objective-C. These alternative languages do not always build cross. Hence this option defaults to "no".

For example:

```
BUILD_OTHER_LANGUAGES=yes
```

**NOTE:** Based upon the version of the compiler being used, it may not be possible to build languages other than C and C++ cross. In many

cases, the language run-time support libraries are not "multilib'ed". Thus the executable code in these libraries will be for the default compiler settings and not necessarily be correct for your CPU model.

`RTEMS` is the directory under tools that contails rtems-DATE.

`ENABLE_RTEMS_POSIX`
is set to "yes" if you want to enable the RTEMS POSIX API support. At this time, this feature is not supported by the UNIX ports of RTEMS and is forced to "no" for those targets. This corresponds to the `configure` option `--enable-posix`.

This must be enabled to support the GNAT/RTEMS run-time.

`ENABLE_RTEMS_ITRON`
is set to "yes" if you want to enable the RTEMS ITRON API support. At this time, this feature is not supported by the UNIX ports of RTEMS and is forced to "no" for those targets. This corresponds to the `configure` option `--enable-itron`.

`ENABLE_RTEMS_MP` is set to "yes" if you want to enable the RTEMS multiprocessing support. This feature is not supported by all RTEMS BSPs and is automatically forced to "no" for those BSPs. This corresponds to the `configure` option `--enable-multiprocessing`.

`ENABLE_RTEMS_CXX` is set to "yes" if you want to build the RTEMS C++ support including the C++ Wrapper for the Classic API. This corresponds to the `configure` option `--enable-cxx`.

`ENABLE_RTEMS_TESTS`
is set to "yes" if you want to build the RTEMS Test Suite. If this is set to "no", then only the Sample Tests will be built. Setting this option to "yes" significantly increases the amount of disk space required to build RTEMS. This corresponds to the `configure` option `--enable-tests`.

`ENABLE_RTEMS_TCPIP`
is set to "yes" if you want to build the RTEMS TCP/IP Stack. If a particular BSP does not support TCP/IP, then this feature is automatically disabled. This corresponds to the `configure` option `--enable-tcpip`.

`ENABLE_RTEMS_NONDEBUG`
is set to "yes" if you want to build RTEMS in a fully optimized state. This corresponds to executing `make` after configuring the source tree.

`ENABLE_RTEMS_DEBUG`
is set to "yes" if you want to build RTEMS in a debug version. When built for debug, RTEMS will include run-time code to perform consistency checks such as heap consistency checks. Although the precise compilation arguments are BSP dependent, the debug version of RTEMS is usually built at a lower optimization level. This is usually done to reduce inlining which can make tracing code execution

difficult. This corresponds to executing `make VARIANT=debug` after configuring the source tree.

`INSTALL_RTEMS` is set to "yes" if you want to install RTEMS after building it. This corresponds to executing `make install` after configuring and building the source tree.

`ENABLE_RTEMS_MAINTAINER_MODE`
is set to "yes" if you want to enabled maintainer mode functionality in the RTEMS Makefile. This is disabled by default and it is not expected that most users will want to enable this. When this option is enabled, the build process may attempt to regenerate files that require tools not required when this option is disabled. This corresponds to the `configure` option `--enable-maintainer-mode`.

## 3.12 Running the bit_ada Script

After the `bit_ada` script has been modified to reflect the local installation, the modified `bit_ada` script is run using the following sequence:

```
cd tools
./bit_ada <target configuration>
```

Where <target configuration> is one of the following:

- hppa1.1
- i386
- m68k
- powerpc
- sh
- sparc

NOTE: The above list of target configurations is the list of RTEMS supported targets. Only a subset of these have been tested with GNAT/RTEMS. For more information, contact your GNAT/RTEMS representative.

The build process can take a while to complete. Many users find it handy to run the build process in the background, capture the output in a file, and monitor the output. This can be done as follows:

```
./bit_ada <target configuration> >bit.log 2>&1 &
tail -f bit.log
```

If no errors are encountered, the `bit_ada` script will conclude by printing messages similar to the following:

```
The src and build-i386-tools subdirectory may now be removed.

Started:  Fri Apr 10 10:14:07 CDT 1998
Finished: Fri Apr 10 12:01:33 CDT 1998
```

If the `bit_ada` script successfully completes, then the GNU C/C++ cross compilation tools are installed.

If the `bit_ada` script does not successfully complete, then investigation will be required to determine the source of the error.

## 3.13 Common Problems

### 3.13.1 Error Message Indicates Invalid Option to Assembler

If a message like this is printed then the new cross compiler is most likely using the native assembler instead of the cross assembler or vice-versa (native compiler using new cross assembler). This can occur for one of the following reasons:

- Binutils Patch Improperly Applied
- Binutils Not Built
- Current Directory is in Your PATH

If you are using binutils 2.9.1 or newer with certain older versions of gcc, they do not agree on what the name of the newly generated cross assembler is. Older binutils called it `as.new` which became `as.new.exe` under Windows. This is not a valid file name, so `as.new` is now called `as-new`. By using the latest released tool versions and RTEMS patches, this problem will be avoided.

If binutils did not successfully build the cross assembler, then the new cross gcc (`xgcc`) used to build the libraries can not find it. Make sure the build of the binutils succeeded.

If you include the current directory in your PATH, then there is a chance that the native compiler will accidentally use the new cross assembler instead of the native one. This usually indicates that "." is before the standard system directories in your PATH. As a general rule, including "." in your PATH is a security risk and should be avoided. Remove "." from your PATH.

NOTE: In some environments, it may be difficult to remove "." completely from your PATH. In this case, make sure that "." is after the system directories containing "as" and "ld".

### 3.13.2 Error Messages Indicating Configuration Problems

If you see error messages like the following,

- cannot configure libiberty
- coff-emulation not found
- etc.

Then it is likely that one or more of your gnu tools is already configured locally in its source tree. You can check for this by searching for the `config.status` file in the various tool source trees. The following command does this for the binutils source:

```
find binutils-2.10 -name config.status -print
```

The solution for this is to execute the command `make distclean` in each of the GNU tools root source directory. This should remove all generated files including Makefiles.

This situation usually occurs when you have previously built the tool source for some non-RTEMS target. The generated configuration specific files are still in the source tree and the include path specified during the RTEMS build accidentally picks up the previous configuration. The include path used is something like this:

```
-I../../binutils-2.10/gcc -I/binutils-2.10/gcc/include -I.
```

Note that the tool source directory is searched before the build directory.

This situation can be avoided entirely by never using the source tree as the build directory – even for

# 4 Building RTEMS

## 4.1 Obtain the RTEMS Source Code

This section provides pointers to the RTEMS source code and Hello World example program. These files should be placed in your `archive` directory.

### RTEMS Snapshot

```
FTP Site:    ftp.rtems.com
Directory:   /pub/rtems/snapshots/current/4.8.2
File:        rtems-ss-DATE.tgz
URL:         ftp://ftp.rtems.com/pub/rtems/snapshots/current/4.8.2/rtems-ss-DATE.t
```

### RTEMS Examples including Hello World

```
FTP Site:    ftp.rtems.com
Directory:   /pub/rtems/snapshots/current/4.8.2
File:        examples-4.8.2.tar.bz2
URL:         ftp://ftp.rtems.com/pub/rtems/snapshots/current/4.8.2/examples-4.8.2.
```

## 4.2 Unarchive the RTEMS Source

Use the following command sequence to unpack the RTEMS source into the tools directory:

```
cd tools
tar xjf ../archive/rtems-ss-DATE.tgz
```

This creates the directory rtems-DATE.

## 4.3 Add <INSTALL_POINT>/bin to Executable PATH

In order to compile RTEMS, you must have the cross compilation toolset in your search path. It is important to have the RTEMS toolset first in your path to ensure that you are using the intended version of all tools. The following command prepends the directory where the tools were installed in a previous step:

```
export PATH=<INSTALL_POINT>/bin:${PATH}
```

NOTE: The above command is in Bourne shell (`sh`) syntax and should work with the Korn (`ksh`) and GNU Bourne Again Shell (`bash`). It will not work with the C Shell (`csh`) or derivatives of the C Shell.

## 4.4 Verifying the Operation of the Cross Toolset

In order to insure that the cross-compiler is invoking the correct subprograms (like `as` and `ld`), one can test assemble a small program. When in verbose mode, `gcc` prints out information showing where it found the subprograms it invokes. In a temporary working directory, place the following function in a file named `f.c`:

```
int f( int x )
{
```

```
  return x + 1;
}
```

Then assemble the file using a command similar to the following:

```
m68k-rtems-gcc -v -S f.c
```

Where `m68k-rtems-gcc` should be changed to match the installed name of your cross compiler. The result of this command will be a sequence of output showing where the cross-compiler searched for and found its subcomponents. Verify that these paths correspond to your <INSTALL_POINT>.

Look at the created file `f.s` and verify that it is in fact for your target processor.

Then try to compile the file `f.c` directly to object code using a command like the following:

```
m68k-rtems-gcc -v -c f.c
```

If this produces messages that indicate the assembly code is not valid, then it is likely that you have fallen victim to one of the problems described in Section 3.13.1 [Error Message Indicates Invalid Option to Assembler], page 14 Don't feel bad about this, one of the most common installation errors is for the cross-compiler not to be able to find the cross assembler and default to using the native `as`. This can result in very confusing error messages.

## 4.5 Building RTEMS for a Specific Target and BSP

This section describes how to configure and build RTEMS so that it is specifically tailored for your BSP and the CPU model it uses. There is currently only one supported method to compile and install RTEMS:

- direct invocation of `configure` and `make`

Direct invocation of `configure` and `make` provides more control and easier recovery from problems when building.

This section describes how to build RTEMS.

### 4.5.1 Using the RTEMS configure Script Directly

Make a build directory under tools and build the RTEMS product in this directory. The `../rtems-DATE/configure` command has numerous command line arguments. These arguments are discussed in detail in documentation that comes with the RTEMS distribution. A full list of these arguments can be obtained by running `../rtems-DATE/configure --help` If you followed the procedure described in the section Section 4.2 [Unarchive the RTEMS Source], page 17, these configuration options can be found in the file tools/rtems-DATE/README.configure.

**NOTE**: The GNAT/RTEMS run-time implementation is based on the POSIX API. Thus the RTEMS configuration for a GNAT/RTEMS environment MUST include the `--enable-posix` flag.

The following shows the command sequence required to configure, compile, and install RTEMS with the POSIX API, FreeBSD TCP/IP, and C++ support disabled. RTEMS will be built to target the `BOARD_SUPPORT_PACKAGE` board.

```
mkdir build-rtems
cd build-rtems
../rtems-DATE/configure --target=<TARGET_CONFIGURATION> \
    --disable-posix --disable-networking --disable-cxx \
    --enable-rtemsbsp=<BOARD_SUPPORT_PACKAGE>\
    --prefix=<INSTALL_POINT>
make all install
```

Where the list of currently supported <TARGET_CONFIGURATION>'s and <BOARD_SUPPORT_PACKAGE>'s can be found in tools/rtems-DATE/README.configure.

<INSTALL_POINT> is typically the installation point for the tools and defaults to `/opt/rtems-4.8`.

BSP is a supported BSP for the selected CPU family. The list of supported BSPs may be found in the file tools/rtems-DATE/README.configure in the RTEMS source tree. If the BSP parameter is not specified, then all supported BSPs for the selected CPU family will be built.

**NOTE:** The POSIX API must be enabled to use GNAT/RTEMS.

**NOTE:** The `make` utility used should be GNU make.

# 5 Building the Sample Application

## 5.1 Unpack the Sample Application

Use the following command to unarchive the sample application:

```
cd tools
tar xzf ../archive/hello_world_ada.tgz
```

## 5.2 Create a BSP Specific Makefile

Provided are example Makefiles for multiple BSPs. Copy one of these to the file Makefile.<BOARD_SUPPORT_PACKAGE> and edit it as appropriate for your local configuration.

Use the <INSTALLATION_POINT> and <BOARD_SUPPORT_PACKAGE> specified when configuring and installing RTEMS.

## 5.3 Build the Sample Application

Use the following command to start the build of the sample application:

```
cd tools/hello_world_ada
make -f Makefile.<BOARD_SUPPORT_PACKAGE>
```

NOTE: GNU make is the preferred `make` utility. Other `make` implementations may work but all testing is done with GNU make.

If the BSP specific modifications to the Makefile were correct and no errors are detected during the sample application build, it is reasonable to assume that the build of the GNAT/RTEMS Cross Compiler Tools for RTEMS and RTEMS itself for the selected host and target combination was done properly.

## 5.4 Application Executable

If the sample application has successfully been build, then the application executable is placed in the following directory:

```
tools/hello_world_ada/o-optimize/<filename>.exe
```

How this executable is downloaded to the target board is very dependent on the BOARD_SUPPORT_PACKAGE selected.

# 6 Building the GNU Debugger

GDB is not currently RTEMS aware. The following configurations have been successfully used with RTEMS applications:

- Sparc Instruction Simulator (SIS)
- PowerPC Instruction Simulator (PSIM)
- DINK32

Other configurations of gdb have successfully been used by RTEMS users but are not documented here.

## 6.1 Unarchive the gdb Distribution

Use the following commands to unarchive the gdb distribution:

```
cd tools
tar xzf ../archive/gdb-4.17.tar.gz
```

The directory gdb-4.17 is created under the tools directory.

## 6.2 Apply GNAT Patch to GDB

No GNAT specific patches are required for gdb 4.17 to support RTEMS Snapshot and gnat 3.13p.

## 6.3 Apply RTEMS Patch to GDB

Apply the patch using the following command sequence:

```
cd tools/gdb-4.17
zcat archive/gdb-4.17-rtems-gnat-3.13p-20000918.diff | patch -p1
```

Check to see if any of these patches have been rejected using the following sequence:

```
cd tools/gdb-4.17
find . -name "*.rej" -print
```

If any files are found with the .rej extension, a patch has been rejected. This should not happen with a good patch file.

To see the files that have been modified use the sequence:

```
cd tools/gdb-4.17
find . -name "*.orig" -print
```

The files that are found, have been modified by the patch file.

## 6.4 GDB with Sparc Instruction Simulation (SIS)

### Make the Build Directory

Create a build directory for the SIS Debugger

```
cd tools
mkdir build-sis
```

### Configure for the Build

Configure the GNU Debugger for the Sparc Instruction Simulator (SIS):

```
cd tools/build-sis
../gdb-4.17/configure --target-sparc-erc32-aout \
    --program-prefix=sparc-rtems- \
    --disable-gdbtk \
    --enable-targets=all \
    --prefix=<INSTALL_POINT_FOR_SIS>
```

Where <INSTALL_POINT_FOR_SIS> is a unique location where the gdb with SIS will be created.

### Make the Debugger

From tools/build-sis execute the following command sequence:

```
make all install
```

NOTE: The `make` utility used should be GNU make.

## 6.5 GDB with PowerPC Instruction Simulator

### Make the Build Directory

Create a build directory for the SIS Debugger

```
cd tools
mkdir build-ppc
```

### Configure for the Build

Configure the GNU Debugger for the PowerPC Instruction Simulator (PSIM):

```
cd tools/build-ppc
../gdb-4.17/configure \
      --target=powerpc-unknown-eabi \
      --program-prefix=powerpc-rtems- \
      --enable-sim-powerpc \
      --enable-sim-timebase \
      --enable-sim-inline \
      --enable-sim-hardware \
      --enable-targets=all \
      --prefix=<INSTALL_POINT_FOR_PPC>
```

Where <INSTALL_POINT_FOR_PPC> is a unique location where the gdb with PSIM will be created.

### Make the Debugger

From tools/build-ppc execute the following command sequence:

```
make all install
```

NOTE: The `make` utility used should be GNU make.

## 6.6 GDB for DINK32

### Make the Build Directory

Create a build directory for the DINK32 Debugger

```
cd tools
mkdir build-dink32
```

### Configure for the Build

Configure the GNU Debugger to communicate with the DINK32 ROM monitor:

```
cd tools/build-dink32
../gdb-4.17/configure --target-powerpc-elf \
    --program-prefix=powerpc-rtems- \
    --enable-targets=all \
    --prefix=<INSTALL_POINT_FOR_DINK32>
```

Where <INSTALL_POINT_FOR_DINK32> is a unique location where the gdb Dink32 will be created.

### Make the Debugger

From tools/build-dink32 execute the following command sequence:

```
make all install
```

NOTE: The `make` utility used should be GNU make.