RTEMS Development Environment Guide

Edition 1, for RTEMS 4.5.0

6 September 2000

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TEXinfo 1999-04-25.15

Any inquiries concerning RTEMS, its related support components, or its documentation should be directed to either:

```
On-Line Applications Research Corporation
4910-L Corporate Drive
Huntsville, AL 35805
VOICE: (256) 722-9985
FAX:   (256) 722-0985
EMAIL: rtems@OARcorp.com
```

1 Introduction

This document describes the RTEMS development environment. Discussions are provided for the following topics:

- the directory structure used by RTEMS,
- usage of the GNU Make utility within the RTEMS development environment,
- sample applications, and
- the RTEMS specific utilities.

RTEMS was designed as a reusable software component. Highly reusable software such as RTEMS is typically distributed in the form of source code without providing any support tools. RTEMS is the foundation for a complex family of facilities including board support packages, device drivers, and support libraries. The RTEMS Development Environment is not a CASE tool. It is a collection of tools designed to reduce the complexity of using and enhancing the RTEMS family. Tools are provided which aid in the management of the development, maintenance, and usage of RTEMS, its run-time support facilities, and applications which utilize the executive.

A key component of the RTEMS development environment is the GNU family of free tools. This is robust set of development and POSIX compatible tools for which source code is freely available. The primary compilers, assemblers, linkers, and make utility used by the RTEMS development team are the GNU tools. They are highly portable supporting a wide variety of host computers and, in the case of the development tools, a wide variety of target processors.

It is recommended that the RTEMS developer become familiar with the RTEMS Development Environment before proceeding with any modifications to the executive source tree. The source code for the executive is very modular and source code is divided amongst directories based upon functionality as well as dependencies on CPU and target board. This organization is aimed at isolating and minimizing non-portable code. This has the immediate result that adding support for a new CPU or target board requires very little "wandering" around the source tree.

2 Directory Structure

The RTEMS directory structure is designed to meet the following requirements:

- encourage development of modular components.
- isolate processor and target dependent code, while allowing as much common source code as possible to be shared across multiple processors and targets.
- allow multiple RTEMS users to perform simultaneous compilation of RTEMS and its support facilities for different processors and targets.

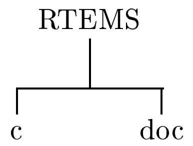
The resulting directory structure has processor and target dependent source files isolated from generic files. When RTEMS is built, object directories and an install point will be automatically created based upon the target BSP selected. The placement of object files based upon the selected BSP name insures that object files are not mixed across CPUs or targets. This in combination with the make files allows the specific compilation options to be tailored for a particular target board. For example, the efficiency of the memory subsystem for a particular target board may be sensitive to the alignment of data structures, while on another target board with the same processor memory may be very limited. For the first target, the options could specify very strict alignment requirements, while on the second the data structures could be "packed" to conserve memory. It is impossible to achieve this degree of flexibility without providing source code.

2.1 Suites

The RTEMS source tree is organized based on the following four variables:

- language,
- target processor,
- target board, and
- compiler vendor (Ada only).

The language may be either C or Ada and there is currently nothing shared between the source trees for these two implementations of RTEMS. The user generally selects the subdirectory for the implementation they are using and ignores that for the other implementation. The only exceptions to this normally occurs when comparing the source code for the two implementations or when porting both to a new CPU or target board. The following shows the top level RTEMS directory structure which includes directories for each language implementation and a language independent source documentation directory. The source documentation directory is currently not supported.



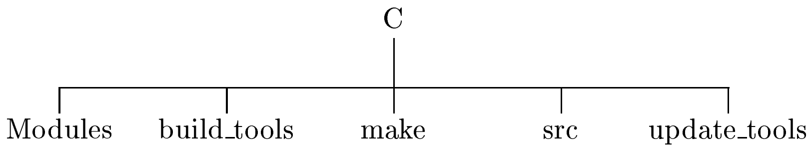
Each of the following sections will describe the contents of the directories in the RTEMS source tree.

2.1.1 C Suites

The following table lists the suites currently included with the C implementation of RTEMS and the directory in which they may be located:

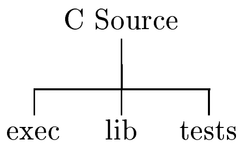
`Support Libraries (BSPs, C library, CPU support)`	$RTEMS_ROOT/c/src/lib
`Single Processor Tests`	$RTEMS_ROOT/c/src/tests/sptests
`Timing Tests`	$RTEMS_ROOT/c/src/tests/tmtests
`Multiprocessor Tests`	$RTEMS_ROOT/c/src/tests/mptests
`Sample Applications`	$RTEMS_ROOT/c/src/tests/samples
`RTEMS Build Tools`	$RTEMS_SRC_BASE/c/build_tools
`Make Support`	$RTEMS_ROOT/c/make

The top level directory structure for the C implementation of RTEMS is as follows:



This directory contains the subdirectories which contain the entire C implementation of the RTEMS executive. The "build-tools" directory contains an assortment of support tools for the RTEMS development environment. Two subdirectories exist under "build-tools" which contain scripts (executables) and source for the support tools. The "make" directory contains configuration files and subdirectories which provide a robust host and cross-target makefile system supporting the building of the executive for numerous application environments. The "update_tools" directory contains utilities which aid in the updating from a previous version to the current version of the RTEMS executive.

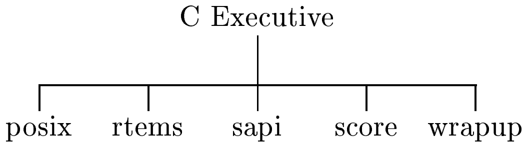
The "src" directory structure for the C implementation of RTEMS is as follows:



This directory contains all source files that comprises the RTEMS executive, supported target board support packages, and the RTEMS Test Suite.

2.1.2 Executive Source Directory

The "exec" directory structure for the C implementation is as follows:



This directory contains a set of subdirectories which contains the source files comprising the executive portion of the RTEMS development environment. At this point the API specific and "supercore" source code files are separated into distinct directory trees. The "rtems" and the "posix" subdirectories contain the C language source files for each module comprising the respective API. Also included in this directory are the subdirectories "sapi" and "score" which are the supercore modules. Within the "score" directory the CPU dependent modules are found.

The "cpu" directory contains a subdirectory for each target CPU supported by the [No value for "RELEASE"] release of the RTEMS executive. Each processor directory contains the CPU dependent code necessary to host RTEMS. The "no_cpu" directory provides a starting point for developing a new port to an unsupported processor. The files contained within the "no_cpu" directory may also be used as a reference for the other ports to specific processors.

2.1.3 Support Library Source Directory

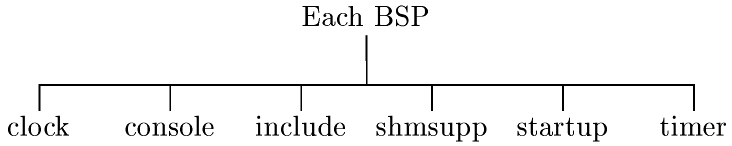
The "lib" directory contains the support libraries and BSPS. Board support packages (BSPs), processor environment start up code, C library support, the FreeBSD TCP/IP stack, common BSP header files, and miscellaneous support functions are provided in the subdirectories. These are combined with the RTEMS executive object to form the single RTEMS library which installed.

The "libbsp" directory contains a directory for each CPU family supported by RTEMS. Beneath each CPU directory is a directory for each BSP for that processor family.

The "libbsp" directory provides all the BSPs provided with this release of the RTEMS executive. The subdirectories are divided, as discussed previously, based on specific processor family, then further breaking down into specific target board environments. The "shmdr" subdirectory provides the implementation of a shared memory driver which supports the multiprocessing portion of the executive. In addition, two starting point subdirectories are provided for reference. The "no_cpu" subdirectory provides a template BSP which can be used to develop a specific BSP for an unsupported target board. The "stubdr" subdirectory provides stubbed out BSPs. These files may aid in preliminary testing of the RTEMS development environment that has been built for no particular target in mind.

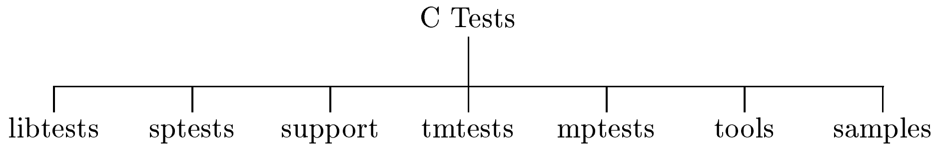
Below each CPU dependent directory is a directory for each target BSP supported in this release.

Each BSP provides the modules which comprise an RTEMS BSP. The modules are separated into the subdirectories "clock", "console", "include", "shmsupp", "startup", and "timer" as shown in the following figure:



2.1.4 Test Suite Source Directory

The "tests" directory structure for the C implementation is as follows:



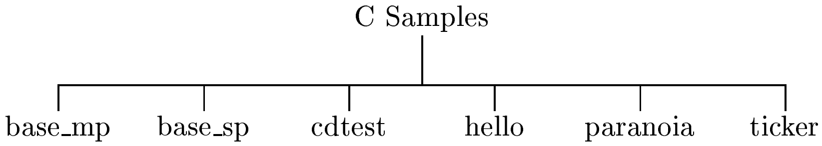
This directory provides the entire RTEMS Test Suite which includes the single processor tests, multiprocessor tests, timing tests, library tests, and sample tests. Additionally, subdirectories for support functions and test related header files are provided.

The "sptests" subdirectory consists of twenty-four tests designed to cover the entire executive code. The "spfatal" test will verify any code associated with the occurrence of a fatal error. Also provided is a test which will determine the size of the RTEMS executive.

The multiprocessor test are provided in "mptests". Fourteen tests are provided in this subdirectory which address two node configurations and cover the multiprocessor code found in RTEMS.

Tests that time each directive and a set of critical executive functions are provided in the "tmtests" subdirectory. Within this subdirectory thirty-one tests are provided along with a subdirectory to contain each targets timing results.

The "samples" directory structure for the C implementation is as follows:



This directory provides sample application tests which aid in the testing a newly built RTEMS environment, a new BSP, or as starting points for the development of an application using the RTEMS executive. A Hello World test is provided in the subdirectory "hello". This test is helpful when testing new versions of RTEMS, BSPs, or modifications to any portion of the RTEMS development environment. The "ticker" subdirectory provides a test for verification of clock chip device drivers of BSPs. A simple single processor test similar to those in the single processor test suite is provided in "base_sp". A simple two node multiprocessor test capable of testing an newly developed MPCI layer is provided in "base_mp". The "cdtest" subdirectory provides a simple C++ application using constructors and destructors. The final sample test is a public domain floating point and math library toolset test is provided in "paranoia".

3 Sample Applications

3.1 Introduction

RTEMS is shipped with the following sample applications:

- Hello World - C and Ada
- Clock Tick - C and Ada
- Base Single Processor - C and Ada
- Base Multiple Processor - C and Ada
- Constructor/Destructor C++ Test - C only if C++ enabled
- Paranoia Floating Point Test - C only

These applications are intended to illustrate the basic format of RTEMS single and multiple processor applications. In addition, these relatively simple applications can be used to test locally developed board support packages and device drivers.

The reader should be familiar with the terms used and material presented in the RTEMS Applications User's Guide.

3.2 Hello World

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/hello
```

It provides a rudimentary test of the BSP start up code and the console output routine. The C version of this sample application uses the printf function from the RTEMS Standard C Library to output messages. The Ada version of this sample use the TEXT_IO package to output the hello messages. The following messages are printed:

```
*** HELLO WORLD TEST ***
Hello World
*** END OF HELLO WORLD TEST ***
```

These messages are printed from the application's single initialization task. If the above messages are not printed correctly, then either the BSP start up code or the console output routine is not operating properly.

3.3 Clock Tick

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/ticker
```

This application is designed as a simple test of the clock tick device driver. In addition, this application also tests the printf function from the RTEMS Standard C Library by using it to output the following messages:

```
*** CLOCK TICK TEST ***
TA1 - tm_get - 09:00:00   12/31/1988
TA2 - tm_get - 09:00:00   12/31/1988
TA3 - tm_get - 09:00:00   12/31/1988
TA1 - tm_get - 09:00:05   12/31/1988
TA1 - tm_get - 09:00:10   12/31/1988
TA2 - tm_get - 09:00:10   12/31/1988
TA1 - tm_get - 09:00:15   12/31/1988
TA3 - tm_get - 09:00:15   12/31/1988
TA1 - tm_get - 09:00:20   12/31/1988
TA2 - tm_get - 09:00:20   12/31/1988
TA1 - tm_get - 09:00:25   12/31/1988
TA1 - tm_get - 09:00:30   12/31/1988
TA2 - tm_get - 09:00:30   12/31/1988
TA3 - tm_get - 09:00:30   12/31/1988
*** END OF CLOCK TICK TEST ***
```

The clock tick sample application utilizes a single initialization task and three copies of the single application task. The initialization task prints the test herald, sets the time and date, and creates and starts the three application tasks before deleting itself. The three application tasks generate the rest of the output. Every five seconds, one or more of the tasks will print the current time obtained via the tm_get directive. The first task, TA1, executes every five seconds, the second task, TA2, every ten seconds, and the third task, TA3, every fifteen seconds. If the time printed does not match the above output, then the clock device driver is not operating properly.

3.4 Base Single Processor Application

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/base_sp
```

It provides a framework from which a single processor RTEMS application can be developed. The use of the task argument is illustrated. This sample application uses the printf function from the RTEMS Standard C Library or TEXT_IO functions when using the Ada version to output the following messages:

```
*** SAMPLE SINGLE PROCESSOR APPLICATION ***
Creating and starting an application task
Application task was invoked with argument (0) and has id of 0x10002
*** END OF SAMPLE SINGLE PROCESSOR APPLICATION ***
```

The first two messages are printed from the application's single initialization task. The final messages are printed from the single application task.

3.5 Base Multiple Processor Application

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/base_mp
```

It provides a framework from which a multiprocessor RTEMS application can be developed. This directory has a subdirectory for each node in the multiprocessor system. The task argument is used to distinguish the node on which the application task is executed. The first node will print the following messages:

```
*** SAMPLE MULTIPROCESSOR APPLICATION ***
Creating and starting an application task
This task was invoked with the node argument (1)
This task has the id of 0x10002
*** END OF SAMPLE MULTIPROCESSOR APPLICATION ***
```

The second node will print the following messages:

```
*** SAMPLE MULTIPROCESSOR APPLICATION ***
Creating and starting an application task
This task was invoked with the node argument (2)
This task has the id of 0x20002
*** END OF SAMPLE MULTIPROCESSOR APPLICATION ***
```

The herald is printed from the application's single initialization task on each node. The final messages are printed from the single application task on each node.

In this sample application, all source code is shared between the nodes except for the node dependent configuration files. These files contains the definition of the node number used in the initialization of the RTEMS Multiprocessor Configuration Table. This file is not shared because the node number field in the RTEMS Multiprocessor Configuration Table must be unique on each node.

3.6 Constructor/Destructor C++ Application

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/cdtest
```

This sample application demonstrates that RTEMS is compatible with C++ applications. It uses constructors, destructor, and I/O stream output in testing these various capabilities. The board support package responsible for this application must support a C++ environment.

This sample application uses the printf function from the RTEMS Standard C Library to output the following messages:

```
Hey I'M in base class constructor number 1 for 0x400010cc.
Hey I'M in base class constructor number 2 for 0x400010d4.
Hey I'M in derived class constructor number 3 for 0x400010d4.
*** CONSTRUCTOR/DESTRUCTOR TEST ***
Hey I'M in base class constructor number 4 for 0x4009ee08.
Hey I'M in base class constructor number 5 for 0x4009ee10.
Hey I'M in base class constructor number 6 for 0x4009ee18.
Hey I'M in base class constructor number 7 for 0x4009ee20.
Hey I'M in derived class constructor number 8 for 0x4009ee20.
Testing a C++ I/O stream
Hey I'M in derived class constructor number 8 for 0x4009ee20.
Derived class - Instantiation order 8
Hey I'M in base class constructor number 7 for 0x4009ee20.
Instantiation order 8
Hey I'M in base class constructor number 6 for 0x4009ee18.
Instantiation order 6
Hey I'M in base class constructor number 5 for 0x4009ee10.
Instantiation order 5
Hey I'M in base class constructor number 4 for 0x4009ee08.
Instantiation order 5
*** END OF CONSTRUCTOR/DESTRUCTOR TEST ***
Hey I'M in base class constructor number 3 for 0x400010d4.
Hey I'M in base class constructor number 2 for 0x400010d4.
Hey I'M in base class constructor number 1 for 0x400010cc.
```

3.7 Paranoia Floating Point Application

This sample application is in the following directory:

```
$RTEMS_SRC_BASE/tests/samples/paranoia
```

This sample application uses a public domain floating point and math library test to verify these capabilities of the RTEMS executive. Deviations between actual and expected results are reported to the screen. This is a very extensive test which tests all mathematical and number conversion functions. Paranoia is also very large and requires a long period of time to run. Problems which commonly prevent this test from executing to completion include stack overflow and FPU exception handlers not installed.

4 RTEMS Specific Utilities

This section describes the additional commands available within the RTEMS Development Environment. Although some of these commands are of general use, most are included to provide some capability necessary to perform a required function in the development of the RTEMS executive, one of its support components, or an RTEMS based application. The commands have been classified into the following categories for clarity:

- C Language Specific Utilities
- Ada Language Specific Utilities

Some of the commands are implemented as C programs. However, most commands are implemented as Bourne shell scripts. Even if the current user has selected a different shell, the scripts will automatically invoke the Bourne shell during their execution lifetime.

The commands are presented in UNIX manual page style for compatibility and convenience. A standard set of paragraph headers were used for all of the command descriptions. If a section contained no data, the paragraph header was omitted to conserve space. Each of the permissible paragraph headers and their contents are described below:

`SYNOPSIS` describes the command syntax

`DESCRIPTION` a full description of the command

`OPTIONS` describes each of the permissible options for the command

`NOTES` lists any special noteworthy comments about the command

`ENVIRONMENT` describes all environment variables utilized by the command

`EXAMPLES` illustrates the use of the command with specific examples

`FILES` provides a list of major files that the command references

`SEE ALSO` lists any relevant commands which can be consulted

Most environment variables referenced by the commands are defined for the RTEMS Development Environment during the login procedure. During login, the user selects a default RTEMS environment through the use of the Modules package. This tool effectively sets the environment variables to provide a consistent development environment for a specific user. Additional environment variables within the RTEMS environment were set by the system administrator during installation. When specifying paths, a command description makes use of these environment variables.

When referencing other commands in the SEE ALSO paragraph, the following notation is used: command(code). Where command is the name of a related command, and code is a section number. Valid section numbers are as follows:

`1` Section 1 of the standard UNIX documentation

`1G` Section 1 of the GNU documentation

`1R` a manual page from this document, the RTEMS Development Environment Guide

For example, ls(1) means see the standard ls command in section 1 of the UNIX documentation. gcc020(1G) means see the description of gcc020 in section 1 of the GNU documentation.

4.1 C Language Specific Utilities

The C language utilities provide a powerful set of tools which combine to allow operations within the RTEMS Development Environment to be consistent and easy to use. Much effort was devoted to providing as close to the standard UNIX and GNU style of operations as possible. Each of these utilities are described in the section below.

4.1.1 packhex - Compress Hexadecimal File

SYNOPSIS

```
packhex <source >destination
```

DESCRIPTION

packhex accepts Intel Hexadecimal or Motorola Srecord on its standard input and attempts to pack as many contiguous bytes as possible into a single hexadecimal record. Many programs output hexadecimal records which are less than 80 bytes long (for human viewing). The overhead required by each unnecessary record is significant and packhex can often reduce the size of the download image by 20%. packhex attempts to output records which are as long as the hexadecimal format allows.

OPTIONS

This command has no options.

EXAMPLES

Assume the current directory contains the Motorola Srecord file download.sr. Then executing the command:

```
packhex <download.sr >packed.sr
```

will generate the file packed.sr which is usually smaller than download.sr.

CREDITS

The source for packhex first appeared in the May 1993 issue of Embedded Systems magazine. The code was downloaded from their BBS. Unfortunately, the author's name was not provided in the listing.

4.1.2 unhex - Convert Hexadecimal File into Binary Equivalent

SYNOPSIS

```
unhex [-valF] [-o file] [file [file ...] ]
```

DESCRIPTION

unhex accepts Intel Hexadecimal, Motorola Srecord, or TI 'B' records and converts them to their binary equivalent. The output may sent to standout or may be placed in a specified file with the -o option. The designated output file may not be an input file. Multiple input files may be specified with their outputs logically concatenated into the output file.

OPTIONS

This command has the following options:

`v`	Verbose
`a base`	First byte of output corresponds with base address
`l`	Linear Output
`o file`	Output File
`F k_bits`	Fill holes in input with 0xFFs up to k_bits * 1024 bits

EXAMPLES

The following command will create a binary equivalent file for the two Motorola S record files in the specified output file binary.bin:

```
unhex -o binary.bin downloadA.sr downloadB.sr
```

4.1.3 size_rtems - report RTEMS size information

SYNOPSIS

```
size_rtems
```

DESCRIPTION

size_rtems analyzes RTEMS and determines all of the critical sizing information which is reported in the related documentation.

EXAMPLES

To generate the RTEMS size report for the currently configured processor, execute the following command:

```
size_rtems
```

Although the actual size information will differ, a report of the following format will be output:

```
     RTEMS SIZE REPORT

CODE  DATA    BSS
==================
MANAGERS: 15988 0       0
CORE    : 4568  0       0
CPU     : 364   0       0
OVERALL : 20556 0       0
MINIMUM : 8752  0       0

init    : 1592  0       0
tasks   : 2440  0       0
intr    : 64    0       0
clock   : 2252  0       0
sem     : 876   0       0
msg     : 1624  0       0
event   : 604   0       0
signal  : 212   0       0
part    : 872   0       0
region  : 844   0       0
dpmem   : 532   0       0
timer   : 424   0       0
io      : 288   0       0
fatal   : 40    0       0
rtmon   : 764   0       0
mp      : 2984  0       0

sem     : 4     0       0
msg     : 4     0       0
event   : 4     0       0
signal  : 4     0       0
part    : 4     0       0
region  : 4     0       0
timer   : 4     0       0
dpmem   : 4     0       0
io      : 4     0       0
rtmon   : 4     0       0
mp      : 8     0       0
```

SEE ALSO

gsize020(1G), gsize386(1G), gsize960(1G)

4.2 Ada Language Specific Utilities

The Ada language utilities provide a powerful set of tools which combine to allow operations within the RTEMS Development Environment to be consistent and easy to use. Much effort was devoted to providing as close to the standard UNIX and GNU style of operations as possible. Each of these utilities are described in the section below.

NOTE: The Ada implementation is not included in this release.

Command and Variable Index

There are currently no Command and Variable Index entries.

Concept Index

There are currently no Concept Index entries.

Table of Contents