RTEMS Development Environment Guide

Edition 4.8.1, for RTEMS 4.8.1

12 August 2008

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

2 Directory Structure 3

2.1 c/ Directory 5
2.1.1 c/src/ Directory 5
2.1.1.1 c/src/lib/libbsp BSP Directory 7
2.2 CPU Kit Directory 7
2.3 testsuites/ Test Suites 10
2.4 Documentation Directory 11

3 Sample Applications 15

3.1 Introduction 15
3.2 Hello World 16
3.3 Clock Tick 17
3.4 Base Single Processor Application 18
3.5 Base Multiple Processor Application 18
3.6 Constructor/Destructor C++ Application 19
3.7 Minimum Size Test 19
3.8 Nanosecond Granularity Application 20
3.9 Paranoia Floating Point Application 21
3.10 Network Loopback Test 21

4 RTEMS Specific Utilities 23

4.1 packhex - Compress Hexadecimal File 24
4.2 unhex - Convert Hexadecimal File into Binary Equivalent 24

Command and Variable Index 27

Concept Index 29

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 target boards.
- allow multiple RTEMS users to perform simultaneous compilation of RTEMS and its support facilities for different processors and targets.

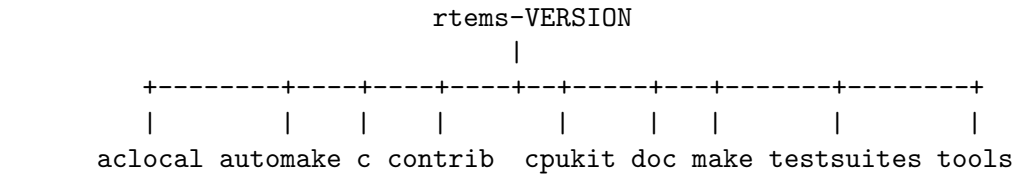
The resulting directory structure has processor and board dependent source files isolated from generic files. When RTEMS is configured and built, object directories and an install point will be automatically created based upon the target CPU family and BSP selected.

The placement of object files based upon the selected BSP name ensures that object files are not mixed across CPUs or targets. This in combination with the makefiles 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.

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

- functionality,
- target processor family,
- target processor model,
- peripherals, and
- target board.

Each of the following sections will describe the contents of the directories in the RTEMS source tree. The top of the tree will be referenced as `${RTEMS_ROOT}` in this discussion.



`${RTEMS_ROOT}/aclocal/`
: This directory contains the custom M4 macros which are available to the various GNU autoconf `configure.ac` scripts throughout the RTEMS source tree. GNU autoconf interprets `configure.ac` files to produce the `configure` files used to tailor RTEMS build for a particular host and target environment. The contents of this directory will not be discussed further in this document.

`${RTEMS_ROOT}/automake/`
: This directory contains the custom GNU automake fragments which are used to support the various `Makefile.am` files throughout the RTEMS source tree. The contents of this directory will not be discussed further in this document.

`${RTEMS_ROOT}/c/`
: This directory is the root of the portions of the RTEMS source tree which must be built tailored for a particular CPU model or BSP. The contents of this directory will be discussed in the Section 2.1 [Directory Structure c/ Directory], page 5 section.

`${RTEMS_ROOT}/contrib/`
: This directory contains contributed support software. Currently this directory contains the RPM specifications for cross-compilers hosted on GNU/Linux that target various operating systems including MinGW, Cygwin, FreeBSD, and Solaris. The cross-compilers produced using these specifications are then used in a Canadian cross build procedure to produce the various RTEMS toolsets on a GNU/Linux host.

  This directory also contains RPM specifications for the prebuilt cross-compilation toolsets provided by the RTEMS project. There are separate subdirectories for each of the components in the RTEMS Cross Compilation Environment unde the `contrib/crossrpms/` directory. This directory is configured, built, and installed separately from the RTEMS executive and tests. This directory will not be discussed further in this document.

`${RTEMS_ROOT}/cpukit/`
: This directory is the root for all of the "multilib'able" portions of RTEMS. This is a GNU way of saying the contents of this directory can be compiled like the C Library (`libc.a`) and the functionality is neither CPU model nor BSP specific. The source code for most RTEMS services reside under this directory. The contents of this directory will be discussed in the Section 2.2 [Directory Structure CPU Kit Directory], page 7 section.

`${RTEMS_ROOT}/doc/`
: This directory is the root for all RTEMS documentation. The source for RTEMS is written in GNU TeXinfo and used to produce HTML, PDF, and "info" files. The RTEMS documentation is configured, built, and installed separately from the RTEMS executive and tests. The contents of this directory will be discussed in the Section 2.4 [Directory Structure Documentation Directory], page 11 section.

`${RTEMS_ROOT}/make/`
: This directory contains files which support the RTEMS Makefile's. From a user's perspective, the most important parts are found in the `custom/` subdirectory. Each ".cfg" file in this directory is associated with a specific BSP and describes the CPU model, compiler flags, and procedure to produce an executable for the target board.

These files are described in detail in the **RTEMS BSP and Device Driver Development Guide** and will not be discussed further in this document.

`${RTEMS_ROOT}/testsuites/`
: This directory contains the test suites for the various RTEMS APIs and support libraries. The contents of this directory are discussed in the Section 2.3 [Directory Structure testsuites/ Test Suites], page 10 section.

`${RTEMS_ROOT}/tools/`
: This directory contains RTEMS specific support utilities which execute on the development host. These utilities are divided into subdirectories based upon whether they are used in the process of building RTEMS and applications, are CPU specific, or are used to assist in updating the RTEMS source tree and applications. The support utilities used in the process of building RTEMS are described in Chapter 4 [RTEMS Specific Utilities], page 23. These are the only components of this subtree that will be discussed in this document.

2.1 c/ Directory

The `${RTEMS_ROOT}/c/` directory was formerly the root directory of all RTEMS source code. At this time, it contains the root directory for only those RTEMS components which must be compiled or linked in a way that is specific to a particular CPU model or board. This directory contains the following subdirectories:

`${RTEMS_ROOT}/c/src/`
: This directory is logically the root for the RTEMS components which are CPU model or board dependent. Thus this directory is the root for the BSPs and the Ada Test Suites as well as CPU model and BSP dependent libraries. The contents of this directory are discussed in the Section 2.1.1 [Directory Structure c/src/ Directory], page 5 section.

2.1.1 c/src/ Directory

As mentioned previously, this directory is logically the root for the RTEMS components which are CPU model or board dependent. The following is a list of the subdirectories in this directory and a description of each.

`${RTEMS_ROOT}/c/src/aclocal/`
: This directory contains the custom M4 macros which are available to the various GNU autoconf `configure.ac` scripts throughout this portion of the RTEMS source tree. GNU autoconf interprets `configure.ac` files to produce the `configure` files used to tailor RTEMS build for a particular host and target environment. The contents of this directory will not be discussed further in this document.

`${RTEMS_ROOT}/c/src/ada/`
: This directory contains the Ada95 language bindings to the RTEMS Classic API.

`${RTEMS_ROOT}/c/src/ada-tests/`
: This directory contains the test suite for the Ada language bindings to the Classic API.

`${RTEMS_ROOT}/c/src/automake/`
: This directory contains files which are "Makefile fragments." They are included as required by the various `Makefile.am` files throughout this portion of the RTEMS source tree.

`${RTEMS_ROOT}/c/src/lib/`
: This directory contains the directories `libbsp/` and `libcpu/` which contain the source code for the Board Support Packages (BSPs) and CPU Model specific source code for RTEMS.

  The `libbsp/` is organized based upon the CPU family and boards BSPs. The contents of `libbsp/` are discussed briefly in Section 2.1.1.1 [Directory Structure c/src/lib/libbsp BSP Directory], page 7 and presented in detail in the **RTEMS BSP and Device Driver Development Guide**.

  The `libcpu/` directory is also organized by CPU family with further divisions based upon CPU model and features that are shared across CPU models such as caching and DMA.

`${RTEMS_ROOT}/c/src/libchip/`
: This directory contains device drivers for various peripheral chips which are designed to be CPU and board dependent. This directory contains a variety of drivers for serial devices, network interface controllers, shared memory and real-time clocks.

`${RTEMS_ROOT}/c/src/librtems++/`
: This directory contains C++ classes which map to the RTEMS Classic API.

`${RTEMS_ROOT}/c/src/make/`
: This directory is used to generate the bulk of the supporting rules files which are installed as part of the Application Makefiles. This file contains settings for various Makefile variables to tailor them to the particular CPU model and BSP configured.

`${RTEMS_ROOT}/c/src/nfsclient/`
: This directory contains a Network File System (NFS) client for RTEMS. With this file system, a user's application can access files on a remote computer.

`${RTEMS_ROOT}/c/src/optman/`
: This directory contains stubs for the RTEMS Classic API Managers which are considered optional and whose use may be explicitly forbidden by an application. All of the directive implementations in this Optional Managers return `E_NOTCONFIGURED`.

`${RTEMS_ROOT}/c/src/support/`
: This directory exists solely to generate the RTEMS version string which includes the RTEMS version, CPU architecture, CPU model, and BSP name.

`${RTEMS_ROOT}/c/src/wrapup/`
: This directory is responsible for taking the individual libraries and objects built in each of the components in the RTEMS source tree and bundling them together to form the single RTEMS library `librtemsbsp.a`. This library contains all BSP and CPU model specific software.

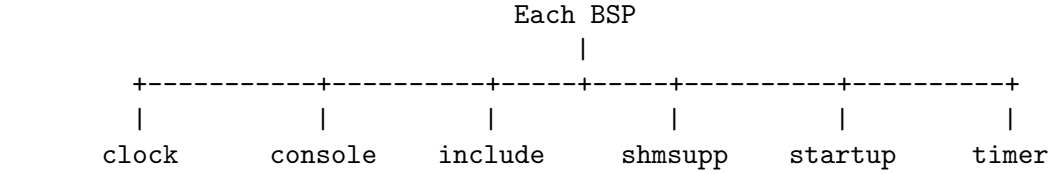
2.1.1.1 c/src/lib/libbsp BSP Directory

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 broken down into specific target board environments. The "no_cpu" subdirectory provides a starting point template BSP which can be used to develop a specific BSP for an unsupported target board. The files in this subdirectory 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:



2.2 CPU Kit Directory

The `cpukit/` directory contains a set of subdirectories which contains the source files comprising the executive portion of the RTEMS development environment as well as portable support libraries such as support for the C Library and filesystems. The API specific and "SuperCore" (e.g. `score/` directory) source code files are separated into distinct directory trees.

The following is a description of each of the subdirectories under `cpukit/`:

`${RTEMS_ROOT}/cpukit/aclocal/`
: This directory contains the custom M4 macros which are available to the various GNU autoconf `configure.ac` scripts throughout the

CPU Kit portion of the RTEMS source tree. GNU autoconf interprets `configure.ac` files to produce the `configure` files used to tailor RTEMS build for a particular host and target environment. The contents of this directory will not be discussed further in this document.

`${RTEMS_ROOT}/cpukit/automake/`
: This directory contains files which are "Makefile fragments." They are included as required by the various `Makefile.am` files throughout the CPU Kit portion of the RTEMS source tree.

`${RTEMS_ROOT}/cpukit/ftpd/`
: This directory contains the RTEMS ftpd server.

`${RTEMS_ROOT}/cpukit/httpd/`
: This directory contains the port of the GoAhead web server to RTEMS.

`${RTEMS_ROOT}/cpukit/include/`
: This directory contains header files which are private to RTEMS and not considered to be owned by any other component in the CPU Kit.

`${RTEMS_ROOT}/cpukit/itron/`
: This directory contains the implementation of the ITRON API.

`${RTEMS_ROOT}/cpukit/libblock/`
: This directory contains support code for using Block Devices such as hard drives, floppies, and CD-ROMs. It includes the generic IO primitives for block device drivers, disk caching support, and a RAM disk block device driver.

`${RTEMS_ROOT}/cpukit/libcsupport/`
: This directory contains the RTEMS specific support routines for the Newlib C Library. This includes what are referred to as system calls and found in section 2 of the traditional UNIX manual. In addition, it contains a thread-safe implementation of the Malloc family of routines as well as BSD and POSIX services not found in Newlib.

`${RTEMS_ROOT}/cpukit/libfs/`
: This directory contains the various non-networked filesystem implementations for RTEMS. It includes the In-Memory FileSystem (IMFS), the mini-IMFS, and FAT filesystems.

`${RTEMS_ROOT}/cpukit/libi2c/`
: This directory contains the RTEMS I2C framework.

`${RTEMS_ROOT}/cpukit/libmd/`
: This directory contains a port of the standard MD5 checksum code.

`${RTEMS_ROOT}/c/src/libmisc/`
: This directory contains support facilities which are RTEMS specific but otherwise unclassified. In general, they do not adhere to a standard API. Among the support facilities in this directory are a

`/dev/null` device driver, the Stack Overflow Checker, a mini-shell, the CPU and rate monotonic period usage monitoring libraries, and a utility to "dump a buffer" in a nicely formatted way similar to many ROM monitors.

`${RTEMS_ROOT}/cpukit/libnetworking/`
: This directory contains the port of the FreeBSD TCP/IP stack to RTEMS.

`${RTEMS_ROOT}/cpukit/librpc/`
: This directory contains the port of the FreeBSD RPC/XDR source to RTEMS.

`${RTEMS_ROOT}/cpukit/posix/`
: This directory contains the RTEMS implementation of the threading portions of the POSIX API.

`${RTEMS_ROOT}/cpukit/pppd/`
: This directory contains a port of the free implementation of the PPPD network protocol.

`${RTEMS_ROOT}/cpukit/rtems/`
: This directory contains the implementation of the Classic API.

`${RTEMS_ROOT}/cpukit/sapi/`
: This directory contains the implementation of RTEMS services which are required but beyond the realm of any standardization efforts. It includes initialization, shutdown, and IO services.

`${RTEMS_ROOT}/cpukit/score/`
: This directory contains the "SuperCore" of RTEMS. All APIs are implemented in terms of SuperCore services. For example, Classic API tasks, POSIX threads, and ITRON tasks are all implemented in terms of SuperCore threads. This provides a common infrastructure and a high degree of interoperability between the APIs. For example, services from all APIs may be used by any task/thread independent of the API used to create it.

  Within the `score/` directory the CPU dependent modules are found. The `score/cpu/` subdirectory contains a subdirectory for each target CPU supported by this 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.

`${RTEMS_ROOT}/cpukit/shttpd/`
: This directory contains the port of the Simple HTTPD web server to RTEMS.

`${RTEMS_ROOT}/cpukit/telnetd/`
: This directory contains the RTEMS telnetd server.

`${RTEMS_ROOT}/cpukit/wrapup/`
: This directory is responsible for taking the individual libraries and objects built in each of the components in the RTEMS CPU Kit source tree and bundling them together to form the single RTEMS library `librtemscpu.a`. This library contains all BSP and CPU model specific software.

`${RTEMS_ROOT}/cpukit/zlib/`
: This directory contains a port of the GNU Zlib compression library to RTEMS.

2.3 testsuites/ Test Suites

This directory provides all of the RTEMS Test Suite except those for the Classic API Ada95 binding This 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 following table lists the test suites currently included with the RTEMS and the directory in which they may be located:

`${RTEMS_ROOT}/testsuites/itrontests/`
: This directory contains the test suite for the RTEMS ITRON API.

`${RTEMS_ROOT}/testsuites/libtests/`
: This directory contains the test suite for the various RTEMS support components.

`${RTEMS_ROOT}/testsuites/mptests/`
: This directory contains the test suite for the multiprocessor support in the Classic API. The tests provided address two node configurations and provide coverage for the multiprocessor code found in RTEMS.

`${RTEMS_ROOT}/testsuites/psxtests/`
: This directory contains the test suite for the RTEMS POSIX API.

`${RTEMS_ROOT}/testsuites/samples/`
: 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. They are discussed in Chapter 3 [Sample Applications], page 15.

`${RTEMS_ROOT}/testsuites/sptests/`
: This directory contains the test suite for the RTEMS Classic API when executing on a single processor. The tests were originally designed to provide near complete test coverage for the entire executive code. With the addition of multiple APIs, this is no longer the case as some SuperCore functionality is not available through the Classic API. Thus some functionality in the SuperCore is only covered by tests in the POSIX API and ITRON API Test Suites.

`${RTEMS_ROOT}/testsuites/support/`
: This directory contains support software and header files for the various test suites.

`${RTEMS_ROOT}/testsuites/tmitrontests/`
: This directory contains the timing test suite for the RTEMS ITRON API.

`${RTEMS_ROOT}/testsuites/tmtests/`
: This directory contains the timing test suite for the RTEMS Classic API. This include tests that benchmark each directive in the Classic API as well as a set of critical SuperCore functions. These tests are important for helping to verify that RTEMS performs as expected on your target hardware. It is not uncommon to discover mistakes in board initialization such as caching being disabled as a side-effect of analyzing the results of these tests.

`${RTEMS_ROOT}/testsuites/tools/`
: This directory contains tools which execute on the development host and aid in executing and evaluating the results of the test suite. The tools `difftest` compares the output of one or more tests with the expected output. If you place the output of all the `tmtests/` in a single file, then the utility `sorttimes` will be able to produce a report organizing the execution times by manager.

2.4 Documentation Directory

This directory contains the source code for all RTEMS documentation in `TexInfo` format as well as utilities used in the generation of the RTEMS documentation set. This source code is used to produce the RTEMS documentation in various formats including PDF, HTML, and PostScript.

`${RTEMS_ROOT}/doc/ada_user/`
: This directory contains the source code for the *RTEMS Applications Ada User's Guide* which documents the Ada95 binding to the Classic API. This manual is produced from from the same source base as the *RTEMS Application C User's Guide*.

`${RTEMS_ROOT}/doc/bsp_howto/`
: This directory contains the source code for the *RTEMS BSP and Device Driver Development Guide*.

`${RTEMS_ROOT}/doc/common/`
: This directory contains the source code for the files which are shared across multiple manuals in the RTEMS Documentation Set. This includes the copyright page as well as the timing tables which can be filled in on a per BSP basis in the CPU supplements.

`${RTEMS_ROOT}/doc/cpu_supplement/`
: This directory contains the source code for the RTEMS CPU Supplement.

`${RTEMS_ROOT}/doc/develenv/`
: This directory contains the source code for the *RTEMS Development Environment Guide*. This is the document you are currently reading.

`${RTEMS_ROOT}/doc/filesystem/`
: This directory contains the source code for the *RTEMS Filesystem Design Guide*. This manual is a continuous work in process as it attempts to capture the design of the interface between system calls and filesystem implementations as well as the information required by those implementing filesystems.

`${RTEMS_ROOT}/doc/images/`
: This directory contains the source code for the graphics used in the HTML version of the RTEMS Documentation.

`${RTEMS_ROOT}/doc/itron3.0/`
: This directory contains the source code for the *RTEMS ITRON 3.0 API User's Guide*.

`${RTEMS_ROOT}/doc/networking/`
: This directory contains the source code for the *RTEMS Network Supplement*.

`${RTEMS_ROOT}/doc/new_chapters/`
: This directory contains the source code for the new documentation components which have not yet been collected into a new manual or merged into an existing document. Currently, this primarily contains draft documentation for some portions of the facilities implemented in `${RTEMS_ROOT}/c/src/libmisc/`.

`${RTEMS_ROOT}/doc/porting/`
: This directory contains the source code for the *RTEMS Porting Guide*.

`${RTEMS_ROOT}/doc/posix1003.1/`
: This directory contains the source code for the *RTEMS POSIX 1003.1 Compliance Guide*.

`${RTEMS_ROOT}/doc/posix_users/`
: This directory contains the source code for the *RTEMS POSIX API User's Guide*. It is important to note that RTEMS' support for POSIX is a combination of functionality provided by RTEMS and the Newlib C Library so some functionality is documented by Newlib.

`${RTEMS_ROOT}/doc/relnotes/`
: This directory contains the source code for a formally release notes document. This has not been used for recent RTEMS releases.

`${RTEMS_ROOT}/doc/started/`
: This directory contains the source code for the *Getting Started with RTEMS for C/C++ Users* manual.

`${RTEMS_ROOT}/doc/started_ada/`
: This directory contains the source code for the *Getting Started with RTEMS for Ada Users* manual.

`${RTEMS_ROOT}/doc/tools/`
: This directory contains the source code for the tools used on the development host to assist in producing the RTEMS Documentation. The most important of these tools is `bmenu` which generates the hierarchical node linking commands based upon chapter, section, and subsection organization.

`${RTEMS_ROOT}/doc/user/`
: This directory contains the source code for the *RTEMS Applications C User's Guide* which documents the Classic API.

3 Sample Applications

3.1 Introduction

The RTEMS source distribution includes a set of sample applications that are located in the `${RTEMS_ROOT}/testsuites/samples/` directory. These applications are intended to illustrate the basic format of RTEMS single and multiple processor applications and the use of some features. In addition, these relatively simple applications can be used to test locally developed board support packages and device drivers as they exercise a critical subset of RTEMS functionality that is often broken in new BSPs.

Some of the following sample applications will be covered in more detail in subsequent sections:

Hello World The RTEMS Hello World test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/hello/`. This test is helpful when testing new RTEMS development environment.

Clock Tick The `${RTEMS_ROOT}/testsuites/samples/ticker/` subdirectory provides a test for verification of clock chip device drivers of BSPs.

Base Single Processor
A simple single processor test similar to those in the single processor test suite is provided in `${RTEMS_ROOT}/testsuites/samples/base_sp/`.

Base Multiple Processor
A simple two node multiprocessor test capable of testing an newly developed MPCI layer is provided in `${RTEMS_ROOT}/testsuites/samples/base_mp/`.

Capture The RTEMS Capture test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/capture/`. This is an interactive test which demonstrates the capabilities of the RTEMS Capture Engine. It includes a few test threads which generate interesting execution patterns. Look at the file `${RTEMS_ROOT}/testsuites/samples/capture/capture.scn` for a sample session.

Constructor/Destructor C++ Test
The `${RTEMS_ROOT}/testsuites/samples/cdtest/` subdirectory provides a simple C++ application using constructors and destructors. It is only built when C++ is enabled and its primary purpose is to demonstrate that global constructors and destructors work. Since this requires that the linker script for your BSP be correct, this is an important test.

File IO The RTEMS File IO test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/fileio/`. This is an interactive test which allows the user to interact with an ATA/IDE device. It will read the partition table and allow the user to dynamically mount

one of the FAT32 partitions it finds. Commands are also provided to write and read files on the disk.

IO Stream The RTEMS IO Stream test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/iostream/`. This test is a simple C++ application which demonstrates that C++ iostreams are functional. This requires that the RTEMS C++ run-time support is functioning properly. This test is only build when C++ is enabled.

Network Loopback Test
The `${RTEMS_ROOT}/testsuites/samples/loopback/` directory contains a sample test that demonstrates the use of sockets and the loopback network device. It does not require the presence of network hardware in order to run. It is only built if RTEMS was configured with networking enabled.

Minimum Size Test The directory `${RTEMS_ROOT}/testsuites/samples/minimum/` contains a simple RTEMS program that results in a non-functional executable. It is intended to show the size of a minimum footprint application based upon the current RTEMS configuration.

Nanoseconds The RTEMS Nanoseconds test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/nsecs/`. This test demonstrates that the BSP has support for nanosecond timestamp granularity. It prints the time of day and uptime multiple times as quickly as possible. It should be possible from the output to determine if your BSP has nanosecond accurate clock support and it is functional.

Paranoia Floating Point Test
The directory `${RTEMS_ROOT}/testsuites/samples/paranoia/` contains the public domain floating point and math library test.

Point-to-Point Protocol Daemon
The RTEMS Point-to-Point Protocol Daemon test is provided in the subdirectory `${RTEMS_ROOT}/testsuites/samples/pppd/`. This test primarily serves as the baseline for a user application using the PPP protocol.

Unlimited Object Allocation
The `${RTEMS_ROOT}/testsuites/samples/unlimited/` directory contains a sample test that demonstrates the use of the *unlimited* object allocation configuration option to RTEMS.

The sample tests are written using the Classic API so the reader should be familiar with the terms used and material presented in the **RTEMS Applications Users Guide**.

3.2 Hello World

This sample application is in the following directory:

```
${RTEMS_ROOT}/testsuites/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 uses 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_ROOT}/testsuites/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_ROOT}/testsuites/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_ROOT}/testsuites/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_ROOT}/testsuites/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 Minimum Size Test

This sample application is in the following directory:

```
${RTEMS_ROOT}/testsuites/samples/minimum/
```

This sample application is designed to produce the minimum code space required for any RTEMS application based upon the current RTEMS configuration and BSP. In many situations, the bulk of this executable consists of hardware and RTEMS initialization, basic infrastructure such as malloc(), and RTEMS and hardware shutdown support.

3.8 Nanosecond Granularity Application

This sample application is in the following directory:

```
${RTEMS_ROOT}/testsuites/samples/nsecs/
```

This sample application exercises the Clock Driver for this BSP and demonstrates its ability to generate accurate timestamps. This application does this by exercising the time subsystem in three ways:

- Obtain Time of Day Twice Back to Back
- Obtain System Up Time Twice Back to Back
- Use System Up Time to Measure Loops

The following is an example of what the output of this test may appear like:

```
*** NANOSECOND CLOCK TEST ***
10 iterations of getting TOD
Start: Sat Mar 24 11:15:00 2007:540000
Stop : Sat Mar 24 11:15:00 2007:549000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:3974000
Stop : Sat Mar 24 11:15:00 2007:3983000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:7510000
Stop : Sat Mar 24 11:15:00 2007:7519000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:11054000
Stop : Sat Mar 24 11:15:00 2007:11063000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:14638000
Stop : Sat Mar 24 11:15:00 2007:14647000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:18301000
Stop : Sat Mar 24 11:15:00 2007:18310000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:21901000
Stop : Sat Mar 24 11:15:00 2007:21910000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:25526000
Stop : Sat Mar 24 11:15:00 2007:25535000 --> 0:9000
Start: Sat Mar 24 11:15:00 2007:29196000
Stop : Sat Mar 24 11:15:00 2007:29206000 --> 0:10000
Start: Sat Mar 24 11:15:00 2007:32826000
Stop : Sat Mar 24 11:15:00 2007:32835000 --> 0:9000

10 iterations of getting Uptime
0:38977000 0:38986000 --> 0:9000
0:40324000 0:40332000 --> 0:8000
0:41636000 0:41645000 --> 0:9000
0:42949000 0:42958000 --> 0:9000
0:44295000 0:44304000 --> 0:9000
0:45608000 0:45617000 --> 0:9000
0:46921000 0:46930000 --> 0:9000
0:48282000 0:48291000 --> 0:9000
0:49595000 0:49603000 --> 0:8000
```

```
0:50908000 0:50917000 --> 0:9000

10 iterations of getting Uptime with different loop values
loop of 10000 0:119488000 0:119704000 --> 0:216000
loop of 20000 0:124028000 0:124463000 --> 0:435000
loop of 30000 0:128567000 0:129220000 --> 0:653000
loop of 40000 0:133097000 0:133964000 --> 0:867000
loop of 50000 0:137643000 0:138728000 --> 0:1085000
loop of 60000 0:142265000 0:143572000 --> 0:1307000
loop of 70000 0:146894000 0:148416000 --> 0:1522000
loop of 80000 0:151519000 0:153260000 --> 0:1741000
loop of 90000 0:156145000 0:158099000 --> 0:1954000
loop of 100000 0:160770000 0:162942000 --> 0:2172000
*** END OF NANOSECOND CLOCK TEST ***
```

3.9 Paranoia Floating Point Application

This sample application is in the following directory:

```
${RTEMS_ROOT}/testsuites/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.

3.10 Network Loopback Test

This sample application is in the following directory:

```
${RTEMS_ROOT}/testsuites/samples/loopback/
```

This sample application uses the network loopback device to demonstrate the use of the RTEMS TCP/IP stack. This sample test illustrates the basic configuration and initialization of the TCP/IP stack as well as simple socket usage.

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.

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 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.

4.2 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
```

Command and Variable Index

There are currently no Command and Variable Index entries.

Concept Index

There are currently no Concept Index entries.