RTEMS Shell User's Guide

Edition 4.10.0, for RTEMS 4.10.0

28 February 2008

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TeXinfo 2009-08-14.15

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

Preface

Real-time embedded systems vary widely based upon their operational and maintenance requirements. Some of these systems provide ways for the user or developer to interact with them. This interaction could be used for operational, diagnostic, or configuration purposes. The capabilities described in this manual are those provided with RTEMS to provide a command line interface for user access. Some of these commands will be familiar as standard POSIX utilities while others are RTEMS specific or helpful in debugging and analyzing an embedded system. As a simple example of the powerful and very familiar capabilities that the RTEMS Shell provides to an application, consider the following example which hints at some of the capabilities available:

```
Welcome to rtems-4.10.0(SPARC/w/FPU/sis)
COPYRIGHT (c) 1989-2011.
On-Line Applications Research Corporation (OAR).

Login into RTEMS

login: rtems
Password:

RTEMS SHELL (Ver.1.0-FRC):/dev/console. Feb 28 2008. 'help' to list commands.
SHLL [/] $ cat /etc/passwd
root:*:0:0:root::/:/bin/sh
rtems:*:1:1:RTEMS Application::/:/bin/sh
tty:!:2:2:tty owner::/:/bin/false
SHLL [/] $ ls /dev
-rwxr-xr-x   1  rtems   root           0 Jan 01 00:00 console
-rwxr-xr-x   1   root   root           0 Jan 01 00:00 console_b
2 files 0 bytes occupied
SHLL [/] $ stackuse
Stack usage by thread
    ID      NAME    LOW          HIGH     CURRENT     AVAILABLE     USED
0x09010001  IDLE 0x023d89a0 - 0x023d99af 0x023d9760      4096        608
0x0a010001  UI1  0x023d9f30 - 0x023daf3f 0x023dad18      4096       1804
0x0a010002  SHLL 0x023db4c0 - 0x023df4cf 0x023de9d0     16384       6204
0xffffffff  INTR 0x023d2760 - 0x023d375f 0x00000000      4080        316
SHLL [/] $ mount -L
File systems: msdos
SHLL [/] $
```

In the above example, the user *rtems* logs into a SPARC based RTEMS system. The first command is `cat /etc/passwd`. This simple command lets us know that this application is running the In Memory File System (IMFS) and that the infrastructure has provided dummy entries for */etc/passwd* and a few other files. The contents of */etc/passwd* let us know that the user could have logged in as `root`. In fact, the `root` user has more permissions than `rtems` who is not allowed to write into the filesystem.

The second command is `ls /dev` which lets us know that RTEMS has POSIX-style device nodes which can be accesses through standard I/O function calls.

The third command executed is the RTEMS specific `stackuse` which gives a report on the stack usage of each thread in the system. Since stack overflows are a common error in deeply embedded systems, this is a surprising simple, yet powerful debugging aid.

Finally, the last command, `mount -L` hints that RTEMS supports a variety of mountable filesystems. With support for MS-DOS FAT on IDE/ATA and Flash devices as well as network-based filesystens such as NFS and TFTP, the standard free RTEMS provides a robuse infrastructure for embedded applications.

This manual describes the RTEMS Shell and its command set. In our terminology, the Shell is just a loop reading user input and turning that input into commands with argument. The Shell provided with RTEMS is a simple command reading loop with limited scripting capabilities. It can be connected to via a standard serial port or connected to the RTEMS `telnetd` server for use across a network.

Each command in the command set is implemented as a single subroutine which has a *main-style* prototype. The commands interpret their arguments and operate upon stdin, stdout, and stderr by default. This allows each command to be invoked independent of the shell.

The described separation of shell from commands from communications mechanism was an important design goal. At one level, the RTEMS Shell is a complete shell environment providing access to multiple POSIX compliant filesystems and TCP/IP stack. The subset of capabilities available is easy to configure and the standard Shell can be logged into from either a serial port or via telnet. But at another level, the Shell is a large set of components which can be integrated into the user's developed command interpreter. In either case, it is trivial to add custom commands to the command set available.

Acknowledgements

The Institute of Electrical and Electronics Engineers, Inc and The Open Group, have given us permission to reprint portions of their documentation.

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2004 Edition, Standard for Information Technology Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright 2001-2004 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html (`http://www.opengroup.org/unix/online.html`).

This notice shall appear on any product containing this material.

1 Configuration and Initialization

1.1 Introduction

This chapter provides information on how the application configures and intializes the RTEMS shell.

1.2 Configuration

The command set available to the application is user configurable. It is configured using a mechanism similar to the `confdefs.h` mechanism used to specify application configuration.

In the simplest case, if the user wishes to configure a command set with all commands available that are neither filesystem management (e.g. mounting, formating, etc.) or network related, then the following is all that is required:

```
#define CONFIGURE_SHELL_COMMANDS_INIT
#define CONFIGURE_SHELL_COMMANDS_ALL

#include <rtems/shellconfig.h>
```

In a slightly more complex example, if the user wishes to include all networking commands as well as support for mounting MS-DOS and NFS filesystems, then the following is all that is required:

```
#define CONFIGURE_SHELL_COMMANDS_INIT
#define CONFIGURE_SHELL_COMMANDS_ALL
#define CONFIGURE_SHELL_MOUNT_MSDOS
#define CONFIGURE_SHELL_MOUNT_NFS

#include <rtems/shellconfig.h>
```

1.2.1 Customizing the Command Set

The user can configure specific command sets by either building up the set from individual commands or starting with a complete set and disabling individual commands. Each command has two configuration macros associated with it.

`CONFIGURE_SHELL_COMMAND_XXX`
: Each command has a constant of this form which is defined when building a command set by individually enabling specific commands.

`CONFIGURE_SHELL_NO_COMMAND_XXX`
: In contrast, each command has a similar command which is defined when the application is configuring a command set by disabling specific commands in the set.

1.2.2 Adding Custom Commands

One of the design goals of the RTEMS Shell was to make it easy for a user to add custom commands specific to their application. We believe this design goal was accomplished. In order to add a custom command, the user is required to do the following:

- Provide a *main-style* function which implements the command. If that command function uses a `getopt` related function to parse arguments, it **MUST** use the reentrant form.
- Provide a command definition structure of type `rtems_shell_cmd_t`.
- Configure that command using the `CONFIGURE_SHELL_USER_COMMANDS` macro.

Custom aliases are configured similarly but the user only provides an alias definition structure of type `rtems_shell_alias_t` and configures the alias via the `CONFIGURE_SHELL_USER_ALIASES` macro.

In the following example, we have implemented a custom command named `usercmd` which simply prints the arguments it was passed. We have also provided an alias for `usercmd` named `userecho`.

```
#include <rtems/shell.h>

int main_usercmd(int argc, char **argv)
{
  int i;
  printf( "UserCommand: argc=%d\n", argc );
  for (i=0 ; i<argc ; i++ )
    printf( "argv[%d]= %s\n", i, argv[i] );
  return 0;
}

rtems_shell_cmd_t Shell_USERCMD_Command = {
  "usercmd",                 /* name */
  "usercmd n1 [n2 [n3...]]", /* usage */
  "user",                    /* topic */
  main_usercmd,              /* command */
  NULL,                      /* alias */
  NULL                       /* next */
};

rtems_shell_alias_t Shell_USERECHO_Alias = {
  "usercmd",                 /* command */
  "userecho"                 /* alias */
};

#define CONFIGURE_SHELL_USER_COMMANDS &Shell_USERCMD_Command
#define CONFIGURE_SHELL_USER_ALIASES &Shell_USERECHO_Alias
#define CONFIGURE_SHELL_COMMANDS_INIT
#define CONFIGURE_SHELL_COMMANDS_ALL
#define CONFIGURE_SHELL_MOUNT_MSDOS

#include <rtems/shellconfig.h>
```

Notice in the above example, that the user wrote the *main* for their command (e.g. `main_usercmd`) which looks much like any other `main()`. They then defined a `rtems_shell_cmd_t` structure named `Shell_USERCMD_Command` which describes that command. This command definition structure is registered into the static command set by defining `CONFIGURE_SHELL_USER_COMMANDS` to `&Shell_USERCMD_Command`.

Similarly, to add the `userecho` alias, the user provides the alias definition structure named `Shell_USERECHO_Alias` and defines `CONFIGURE_SHELL_USER_ALIASES` to configure the alias.

The user can configure any number of commands and aliases in this manner.

1.3 Initialization

The shell may be easily attached to a serial port or to the `telnetd` server. This section describes how that is accomplished.

1.3.1 Attached to a Serial Port

Starting the shell attached to the console or a serial port is very simple. The user invokes `rtems_shell_init` with parameters to indicate the characteristics of the task that will be executing the shell including name, stack size, and priority. The user also specifies the device that the shell is to be attached to.

This example is taken from the `fileio` sample test. This shell portion of this test can be run on any target which provides a console with input and output capabilities. It does not include any commands which cannot be supported on all BSPs. The source code for this test is in `testsuites/samples/fileio` with the shell configuration in the `init.c` file.

```
#include <rtems/shell.h>

void start_shell(void)
{
  printf(" =========================\n");
  printf(" starting shell\n");
  printf(" =========================\n");
  rtems_shell_init(
    "SHLL",                       /* task name */
    RTEMS_MINIMUM_STACK_SIZE * 4, /* task stack size */
    100,                          /* task priority */
    "/dev/console",               /* device name */
    false,                        /* run forever */
    true,                         /* wait for shell to terminate */
    rtems_shell_login_check       /* login check function,
                                     use NULL to disable a login check */
  );
}
```

In the above example, the call to `rtems_shell_init` spawns a task to run the RTEMS Shell attached to `/dev/console` and executing at priority 100. The caller suspends itself and lets the shell take over the console device. When the shell is exited by the user, then control returns to the caller.

1.3.2 Attached to a Socket

TBD

1.4 Functions

This section describes the Shell related C functions which are publicly available related to initialization and configuration.

1.4.1 rtems_shell_init - initialize the shell

CALLING SEQUENCE:

```
rtems_status_code rtems_shell_init(
  const char          *task_name,
  size_t               task_stacksize,
  rtems_task_priority  task_priority,
  const char          *devname,
  bool                 forever,
  bool                 wait,
  rtems_login_check    login_check
);
```

DIRECTIVE STATUS CODES:

`RTEMS_SUCCESSFUL` - Shell task spawned successfully
others - to indicate a failure condition

DESCRIPTION:

This service creates a task with the specified characteristics to run the RTEMS Shell attached to the specified `devname`.

NOTES:

This method invokes the `rtems_task_create` and `rtems_task_start` directives and as such may return any status code that those directives may return.

2 General Commands

2.1 Introduction

The RTEMS shell has the following general commands:

- `alias` - Add alias for an existing command
- `date` - Print or set current date and time
- `echo` - Produce message in a shell script
- `sleep` - Delay for a specified amount of time
- `id` - show uid gid euid and egid
- `tty` - show ttyname
- `whoami` - print effective user id
- `getenv` - print environment variable
- `setenv` - set environment variable
- `unsetenv` - unset environment variable
- `time` - time command execution
- `logoff` - logoff from the system
- `rtc` - RTC driver configuration
- `exit` - alias for logoff command

2.2 Commands

This section details the General Commands available. A subsection is dedicated to each of the commands and describes the behavior and configuration of that command as well as providing an example usage.

2.2.1 alias - add alias for an existing command

SYNOPSYS:

```
alias oldCommand newCommand
```

DESCRIPTION:

This command adds an alternate name for an existing command to the command set.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `alias`:

```
SHLL [/] $ me
shell:me command not found
SHLL [/] $ alias whoami me
SHLL [/] $ me
rtems
SHLL [/] $ whoami
rtems
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_ALIAS` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_ALIAS` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `alias` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_alias(
  int    argc,
  char **argv
);
```

The configuration structure for the `alias` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_ALIAS_Command;
```

2.2.2 date - print or set current date and time

SYNOPSYS:

```
date
date DATE TIME
```

DESCRIPTION:

This command operates one of two modes. When invoked with no arguments, it prints the current date and time. When invoked with both `date` and `time` arguments, it sets the current time.

The `date` is specified in `YYYY-MM-DD` format. The `time` is specified in `HH:MM:SS` format.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This comm

EXAMPLES:

The following is an example of how to use `date`:

```
SHLL [/] $ date
Fri Jan  1 00:00:09 1988
SHLL [/] $ date 2008-02-29 06:45:32
SHLL [/] $ date
Fri Feb 29 06:45:35 2008
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DATE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DATE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `date` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_date(
  int    argc,
  char **argv
);
```

The configuration structure for the `date` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DATE_Command;
```

2.2.3 echo - produce message in a shell script

SYNOPSYS:

```
echo [-n | -e] args ...
```

DESCRIPTION:

echo prints its arguments on the standard output, separated by spaces. Unless the **-n** option is present, a newline is output following the arguments. The **-e** option causes echo to treat the escape sequences specially, as described in the following paragraph. The **-e** option is the default, and is provided solely for compatibility with other systems. Only one of the options **-n** and **-e** may be given.

If any of the following sequences of characters is encountered during output, the sequence is not output. Instead, the specified action is performed:

\b A backspace character is output.

\c Subsequent output is suppressed. This is normally used at the end of the last argument to suppress the trailing newline that echo would otherwise output.

\f Output a form feed.

\n Output a newline character.

\r Output a carriage return.

\t Output a (horizontal) tab character.

\v Output a vertical tab.

\0digits Output the character whose value is given by zero to three digits. If there are zero digits, a nul character is output.

**** Output a backslash.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The octal character escape mechanism (\0digits) differs from the C lan- guage mechanism.

There is no way to force `echo` to treat its arguments literally, rather than interpreting them as options and escape sequences.

EXAMPLES:

The following is an example of how to use `echo`:

```
SHLL [/] $ echo a b c
a b c
SHLL [/] $ echo
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_ECHO` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_ECHO` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `echo` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_echo(
  int    argc,
  char **argv
);
```

The configuration structure for the `echo` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_ECHO_Command;
```

ORIGIN:

The implementation and portions of the documentation for this command are from NetBSD 4.0.

2.2.4 sleep - delay for a specified amount of time

SYNOPSYS:

```
sleep seconds
sleep seconds nanoseconds
```

DESCRIPTION:

This command causes the task executing the shell to block for the specified number of `seconds` and `nanoseconds`.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This command is implemented using the `nanosleep()` method.

The command line interface is similar to the `sleep` command found on POSIX systems but the addition of the `nanoseconds` parameter allows fine grained delays in shell scripts without adding another command such as `usleep`.

EXAMPLES:

The following is an example of how to use `sleep`:

```
SHLL [/] $ sleep 10
SHLL [/] $ sleep 0 5000000
```

It is not clear from the above but there is a ten second pause after executing the first command before the prompt is printed. The second command completes very quickly from a human perspective and there is no noticeable delay in the prompt being printed.

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_SLEEP` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_SLEEP` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `sleep` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_sleep(
  int    argc,
  char **argv
);
```

The configuration structure for the `sleep` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_SLEEP_Command;
```

2.2.5 id - show uid gid euid and egid

SYNOPSYS:

```
id
```

DESCRIPTION:

This command prints the user identity. This includes the user id (uid), group id (gid), effective user id (euid), and effective group id (egid).

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

Remember there is only one POSIX process in a single processor RTEMS application. Each thread may have its own user identity and that identity is used by the filesystem to enforce permissions.

EXAMPLES:

The first example of the `id` command is from a session logged in as the normal user `rtems`:

```
SHLL [/] # id
uid=1(rtems),gid=1(rtems),euid=1(rtems),egid=1(rtems)
```

The second example of the `id` command is from a session logged in as the `root` user:

```
SHLL [/] # id
uid=0(root),gid=0(root),euid=0(root),egid=0(root)
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_ID` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_ID` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `id` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_id(
  int    argc,
  char **argv
);
```

The configuration structure for the `id` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_ID_Command;
```

2.2.6 tty - show ttyname

SYNOPSYS:

```
tty
```

DESCRIPTION:

This command prints the file name of the device connected to standard input.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `tty`:

```
SHLL [/] $ tty
/dev/console
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_TTY` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_TTY` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `tty` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_tty(
  int    argc,
  char **argv
);
```

The configuration structure for the `tty` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_TTY_Command;
```

2.2.7 whoami - print effective user id

SYNOPSYS:

```
whoami
```

DESCRIPTION:

This command displays the user name associated with the current effective user id.

EXIT STATUS:

This command always succeeds.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `whoami`:

```
SHLL [/] $ whoami
rtems
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_WHOAMI` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_WHOAMI` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `whoami` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_whoami(
  int    argc,
  char **argv
);
```

The configuration structure for the `whoami` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_WHOAMI_Command;
```

2.2.8 getenv - print environment variable

SYNOPSYS:

```
getenv variable
```

DESCRIPTION:

This command is used to display the value of a `variable` in the set of environment variables.

EXIT STATUS:

This command will return 1 and print a diagnostic message if a failure occurs.

NOTES:

The entire RTEMS application shares a single set of environment variables.

EXAMPLES:

The following is an example of how to use `getenv`:

```
SHLL [/] $ getenv BASEPATH
/mnt/hda1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_GETENV` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_GETENV` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `getenv` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_getenv(
  int    argc,
  char **argv
);
```

The configuration structure for the `getenv` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_GETENV_Command;
```

2.2.9 setenv - set environment variable

SYNOPSYS:

```
setenv variable [value]
```

DESCRIPTION:

This command is used to add a new `variable` to the set of environment variables or to modify the variable of an already existing `variable`. If the `value` is not provided, the `variable` will be set to the empty string.

EXIT STATUS:

This command will return 1 and print a diagnostic message if a failure occurs.

NOTES:

The entire RTEMS application shares a single set of environment variables.

EXAMPLES:

The following is an example of how to use `setenv`:

```
SHLL [/] $ setenv BASEPATH /mnt/hda1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_SETENV` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_SETENV` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `setenv` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_setenv(
  int    argc,
  char **argv
);
```

The configuration structure for the `setenv` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_SETENV_Command;
```

2.2.10 unsetenv - unset environment variable

SYNOPSYS:

```
unsetenv variable
```

DESCRIPTION:

This command is remove to a `variable` from the set of environment variables.

EXIT STATUS:

This command will return 1 and print a diagnostic message if a failure occurs.

NOTES:

The entire RTEMS application shares a single set of environment variables.

EXAMPLES:

The following is an example of how to use `unsetenv`:

```
SHLL [/] $ unsetenv BASEPATH
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_UNSETENV` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_UNSETENV` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `unsetenv` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_unsetenv(
  int    argc,
  char **argv
);
```

The configuration structure for the `unsetenv` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_UNSETENV_Command;
```

2.2.11 time - time command execution

SYNOPSYS:

```
time command [argument ...]
```

DESCRIPTION:

The time command executes and times a command. After the command finishes, time writes the total time elapsed. Times are reported in seconds.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

None.

EXAMPLES:

The following is an example of how to use `time`:

```
SHLL [/] $ time cp -r /nfs/directory /c
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_TIME` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_TIME` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `time` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_time(
  int    argc,
  char **argv
);
```

The configuration structure for the `time` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_TIME_Command;
```

2.2.12 logoff - logoff from the system

SYNOPSYS:

```
logoff
```

DESCRIPTION:

This command logs the user out of the shell.

EXIT STATUS:

This command does not return.

NOTES:

The system behavior when the shell is exited depends upon how the shell was initiated. The typical behavior is that a login prompt will be displayed for the next login attempt or that the connection will be dropped by the RTEMS system.

EXAMPLES:

The following is an example of how to use `logoff`:

```
SHLL [/] $ logoff
logoff from the system...
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_LOGOFF` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_LOGOFF` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `logoff` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_logoff(
  int    argc,
  char **argv
);
```

The configuration structure for the `logoff` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_LOGOFF_Command;
```

2.2.13 rtc - RTC driver configuration

SYNOPSYS:

```
rtc
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_RTC` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_RTC` when all shell commands have been configured.

2.2.14 exit - exit the shell

SYNOPSYS:

```
exit
```

DESCRIPTION:

This command causes the shell interpreter to `exit`.

EXIT STATUS:

This command does not return.

NOTES:

In contrast to Section 2.2.12 [General Commands logoff - logoff from the system], page 20, this command is built into the shell interpreter loop.

EXAMPLES:

The following is an example of how to use `exit`:

```
SHLL [/] $ exit
Shell exiting
```

CONFIGURATION:

This command is always present and cannot be disabled.

PROGRAMMING INFORMATION:

The `exit` is implemented directly in the shell interpreter. There is no C routine associated with it.

3 File and Directory Commands

3.1 Introduction

The RTEMS shell has the following file and directory commands:

- `umask` - Set file mode creation mask
- `cp` - copy files
- `mv` - move files
- `pwd` - print work directory
- `ls` - list files in the directory
- `chdir` - change the current directory
- `mkdir` - create a directory
- `rmdir` - remove empty directories
- `ln` - make links
- `mknod` - make device special file
- `chroot` - change the root directory
- `chmod` - change permissions of a file
- `cat` - display file contents
- `msdosfmt` - format disk
- `rm` - remove files
- `mount` - mount disk
- `unmount` - unmount disk
- `blksync` - sync the block driver
- `dd` - format disks
- `hexdump` - format disks
- `fdisk` - format disks
- `dir` - alias for ls
- `mkrfs` - format RFS file system
- `cd` - alias for chdir

3.2 Commands

This section details the File and Directory Commands available. A subsection is dedicated to each of the commands and describes the behavior and configuration of that command as well as providing an example usage.

3.2.1 umask - set file mode creation mask

SYNOPSYS:

```
umask [new_umask]
```

DESCRIPTION:

This command sets the user file creation mask to `new_umask`. The argument `new_umask` may be octal, hexadecimal, or decimal.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This command does not currently support symbolic mode masks.

EXAMPLES:

The following is an example of how to use `umask`:

```
SHLL [/] $ umask
022
SHLL [/] $ umask 0666
0666
SHLL [/] $ umask
0666
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_UMASK` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_UMASK` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `umask` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_umask(
  int    argc,
  char **argv
);
```

The configuration structure for the `umask` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_UMASK_Command;
```

3.2.2 cp - copy files

SYNOPSYS:

```
cp [-R [-H | -L | -P]] [-f | -i] [-pv] src target
cp [-R [-H | -L] ] [-f | -i] [-NpPv] source_file ... target_directory
```

DESCRIPTION:

In the first synopsis form, the cp utility copies the contents of the source_file to the target_file. In the second synopsis form, the contents of each named source_file is copied to the destination target_directory. The names of the files themselves are not changed. If cp detects an attempt to copy a file to itself, the copy will fail.

The following options are available:

-f For each existing destination pathname, attempt to overwrite it. If permissions do not allow copy to succeed, remove it and create a new file, without prompting for confirmation. (The -i option is ignored if the -f option is specified.)

-H If the -R option is specified, symbolic links on the command line are followed. (Symbolic links encountered in the tree traversal are not followed.)

-i Causes cp to write a prompt to the standard error output before copying a file that would overwrite an existing file. If the response from the standard input begins with the character 'y', the file copy is attempted.

-L If the -R option is specified, all symbolic links are followed.

-N When used with -p, do not copy file flags.

-P No symbolic links are followed.

-p Causes cp to preserve in the copy as many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions.

If the user ID and group ID cannot be preserved, no error message is displayed and the exit value is not altered.

If the source file has its set user ID bit on and the user ID cannot be preserved, the set user ID bit is not preserved in the copy's permissions. If the source file has its set group ID bit on and the group ID cannot be preserved, the set group ID bit is not preserved in the copy's permissions. If the source file has both its set user ID and set group ID bits on, and either the user ID or group ID cannot be preserved, neither the set user ID or set group ID bits are preserved in the copy's permissions.

-R If source_file designates a directory, cp copies the directory and the entire subtree connected at that point. This option also causes symbolic links to be copied, rather than indirected through, and for cp to create special files rather than copying them as normal files. Created directories have the same mode as the corresponding source directory, unmodified by the process's umask.

-v Cause cp to be verbose, showing files as they are copied.

For each destination file that already exists, its contents are overwritten if permissions allow, but its mode, user ID, and group ID are unchanged.

In the second synopsis form, target_directory must exist unless there is only one named source_file which is a directory and the -R flag is specified.

If the destination file does not exist, the mode of the source file is used as modified by the file mode creation mask (umask, see csh(1)). If the source file has its set user ID bit on, that bit is removed unless both the source file and the destination file are owned by the same user. If the source file has its set group ID bit on, that bit is removed unless both the source file and the destination file are in the same group and the user is a member of that group. If both the set user ID and set group ID bits are set, all of the above conditions must be fulfilled or both bits are removed.

Appropriate permissions are required for file creation or overwriting.

Symbolic links are always followed unless the -R flag is set, in which case symbolic links are not followed, by default. The -H or -L flags (in conjunction with the -R flag), as well as the -P flag cause symbolic links to be followed as described above. The -H and -L options are ignored unless the -R option is specified. In addition, these options override eachsubhedading other and the command's actions are determined by the last one specified.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `cp` to copy a file to a new name in the current directory:

```
SHLL [/] # cat joel
cat: joel: No such file or directory
SHLL [/] # cp etc/passwd joel
SHLL [/] # cat joel
root:*:0:0:root::/:/bin/sh
rtems:*:1:1:RTEMS Application::/:/bin/sh
tty:!:2:2:tty owner::/:/bin/false
SHLL [/] # ls
drwxr-xr-x   1   root   root         536 Jan 01 00:00 dev/
drwxr-xr-x   1   root   root        1072 Jan 01 00:00 etc/
-rw-r--r--   1   root   root         102 Jan 01 00:00 joel
3 files 1710 bytes occupied
```

The following is an example of how to use `cp` to copy one or more files to a destination directory and use the same `basename` in the destination directory:

```
SHLL [/] # mkdir tmp
SHLL [/] # ls tmp
0 files 0 bytes occupied
SHLL [/] # cp /etc/passwd tmp
SHLL [/] # ls /tmp
-rw-r--r--   1   root   root         102 Jan 01 00:01 passwd
1 files 102 bytes occupied
SHLL [/] # cp /etc/passwd /etc/group /tmp
SHLL [/] # ls /tmp
-rw-r--r--   1   root   root         102 Jan 01 00:01 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:01 group
2 files 144 bytes occupied
SHLL [/] #
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CP` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CP` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `cp` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_cp(
  int    argc,
  char **argv
);
```

The configuration structure for the `cp` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CP_Command;
```

ORIGIN:

The implementation and portions of the documentation for this command are from NetBSD 4.0.

3.2.3 mv - move files

SYNOPSYS:

```
mv [-fiv] source_file target_file
mv [-fiv] source_file... target_file
```

DESCRIPTION:

In its first form, the mv utility renames the file named by the source operand to the destination path named by the target operand. This form is assumed when the last operand does not name an already existing directory.

In its second form, mv moves each file named by a source operand to a destination file in the existing directory named by the directory operand. The destination path for each operand is the pathname produced by the concatenation of the last operand, a slash, and the final pathname component of the named file.

The following options are available:

-f Do not prompt for confirmation before overwriting the destination path.

-i Causes mv to write a prompt to standard error before moving a file that would overwrite an existing file. If the response from the standard input begins with the character 'y', the move is attempted.

-v Cause mv to be verbose, showing files as they are processed.

The last of any -f or -i options is the one which affects mv's behavior.

It is an error for any of the source operands to specify a nonexistent file or directory.

It is an error for the source operand to specify a directory if the target exists and is not a directory.

If the destination path does not have a mode which permits writing, mv prompts the user for confirmation as specified for the -i option.

Should the **rename** call fail because source and target are on different file systems, `mv` will remove the destination file, copy the source file to the destination, and then remove the source. The effect is roughly equivalent to:

```
rm -f destination_path && \
cp -PRp source_file destination_path && \
rm -rf source_file
```

EXIT STATUS:

The `mv` utility exits 0 on success, and >0 if an error occurs.

NOTES:

NONE

EXAMPLES:

```
SHLL [/] mv /dev/console /dev/con1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MV` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MV` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mv` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mv(
  int    argc,
  char **argv
);
```

The configuration structure for the `mv` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MV_Command;
```

ORIGIN:

The implementation and portions of the documentation for this command are from NetBSD 4.0.

3.2.4 pwd - print work directory

SYNOPSYS:

```
pwd
```

DESCRIPTION:

This command prints the fully qualified filename of the current working directory.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `pwd`:

```
SHLL [/] $ pwd
/
SHLL [/] $ cd dev
SHLL [/dev] $ pwd
/dev
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_PWD` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_PWD` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `pwd` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_pwd(
  int    argc,
  char **argv
);
```

The configuration structure for the `pwd` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_PWD_Command;
```

3.2.5 ls - list files in the directory

SYNOPSYS:

```
ls [dir]
```

DESCRIPTION:

This command displays the contents of the specified directory. If no arguments are given, then it displays the contents of the current working directory.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This command currently does not display information on a set of files like the POSIX ls(1). It only displays the contents of entire directories.

EXAMPLES:

The following is an example of how to use `ls`:

```
SHLL [/] $ ls
drwxr-xr-x   1   root   root         536 Jan 01 00:00 dev/
drwxr-xr-x   1   root   root        1072 Jan 01 00:00 etc/
2 files 1608 bytes occupied
SHLL [/] $ ls etc
-rw-r--r--   1   root   root         102 Jan 01 00:00 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:00 group
-rw-r--r--   1   root   root          30 Jan 01 00:00 issue
-rw-r--r--   1   root   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
SHLL [/] $ ls dev etc
-rwxr-xr-x   1  rtems   root           0 Jan 01 00:00 console
-rwxr-xr-x   1   root   root           0 Jan 01 00:00 console_b
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_LS` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_LS` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `ls` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_ls(
  int    argc,
  char **argv
);
```

The configuration structure for the `ls` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_LS_Command;
```

3.2.6 chdir - change the current directory

SYNOPSYS:

```
chdir [dir]
```

DESCRIPTION:

This command is used to change the current working directory to the specified directory. If no arguments are given, the current working directory will be changed to `/`.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `chdir`:

```
SHLL [/] $ pwd
/
SHLL [/] $ chdir etc
SHLL [/etc] $ pwd
/etc
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CHDIR` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CHDIR` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `chdir` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_chdir(
  int    argc,
  char **argv
);
```

The configuration structure for the `chdir` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CHDIR_Command;
```

3.2.7 mkdir - create a directory

SYNOPSYS:

```
mkdir  dir [dir1 .. dirN]
```

DESCRIPTION:

This command creates the set of directories in the order they are specified on the command line. If an error is encountered making one of the directories, the command will continue to attempt to create the remaining directories on the command line.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

If this command is invoked with no arguments, nothing occurs.

The user must have sufficient permissions to create the directory. For the `fileio` test provided with RTEMS, this means the user must login as `root` not `rtems`.

EXAMPLES:

The following is an example of how to use `mkdir`:

```
SHLL [/] # ls
drwxr-xr-x   1   root   root         536 Jan 01 00:00 dev/
drwxr-xr-x   1   root   root        1072 Jan 01 00:00 etc/
2 files 1608 bytes occupied
SHLL [/] # mkdir joel
SHLL [/] # ls joel
0 files 0 bytes occupied
SHLL [/] # cp etc/passwd joel
SHLL [/] # ls joel
-rw-r--r--   1   root   root         102 Jan 01 00:02 passwd
1 files 102 bytes occupied
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MKDIR` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MKDIR` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mkdir` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mkdir(
  int    argc,
  char **argv
);
```

The configuration structure for the `mkdir` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MKDIR_Command;
```

3.2.8 rmdir - remove empty directories

SYNOPSYS:

```
rmdir  [dir1 .. dirN]
```

DESCRIPTION:

This command removes the specified set of directories. If no directories are provided on the command line, no actions are taken.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This command is a implemented using the `rmdir(2)` system call and all reasons that call may fail apply to this command.

EXAMPLES:

The following is an example of how to use `rmdir`:

```
SHLL [/] # mkdir joeldir
SHLL [/] # rmdir joeldir
SHLL [/] # ls joeldir
joeldir: No such file or directory.
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_RMDIR` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_RMDIR` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `rmdir` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_rmdir(
  int    argc,
  char **argv
);
```

The configuration structure for the `rmdir` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_RMDIR_Command;
```

3.2.9 ln - make links

SYNOPSYS:

```
ln [-fhinsv] source_file [target_file]
ln [-fhinsv] source_file ... target_dir
```

DESCRIPTION:

The ln utility creates a new directory entry (linked file) which has the same modes as the original file. It is useful for maintaining multiple copies of a file in many places at once without using up storage for the "copies"; instead, a link "points" to the original copy. There are two types of links; hard links and symbolic links. How a link "points" to a file is one of the differences between a hard or symbolic link.

The options are as follows:

-f Unlink any already existing file, permitting the link to occur.

-h If the target_file or target_dir is a symbolic link, do not follow it. This is most useful with the -f option, to replace a symlink which may point to a directory.

-i Cause ln to write a prompt to standard error if the target file exists. If the response from the standard input begins with the character 'y' or 'Y', then unlink the target file so that the link may occur. Otherwise, do not attempt the link. (The -i option overrides any previous -f options.)

-n Same as -h, for compatibility with other ln implementations.

-s Create a symbolic link.

-v Cause ln to be verbose, showing files as they are processed.

By default ln makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not normally refer to directories and may not span file systems.

A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open* operation is performed on the link. A *stat* on a symbolic link will return the linked-to file; an *lstat* must be done to obtain information about the link. The *readlink* call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

Given one or two arguments, ln creates a link to an existing file source_file. If target_file is given, the link has that name; target_file may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of source_file.

Given more than two arguments, ln makes links in target_dir to all the named source files. The links made will have the same name as the files being linked to.

EXIT STATUS:

The `ln` utility exits 0 on success, and >0 if an error occurs.

NOTES:

NONE

EXAMPLES:

```
SHLL [/] ln -s /dev/console /dev/con1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_LN` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_LN` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `ln` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_ln(
  int    argc,
  char **argv
);
```

The configuration structure for the `ln` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_LN_Command;
```

ORIGIN:

The implementation and portions of the documentation for this command are from NetBSD 4.0.

3.2.10 mknod - make device special file

SYNOPSYS:

```
mknod [-rR] [-F fmt] [-g gid] [-m mode] [-u uid] name [c | b]
      [driver | major] minor
mknod [-rR] [-F fmt] [-g gid] [-m mode] [-u uid] name [c | b]
      major unit subunit
mknod [-rR] [-g gid] [-m mode] [-u uid] name [c | b] number
mknod [-rR] [-g gid] [-m mode] [-u uid] name p
```

DESCRIPTION:

The mknod command creates device special files, or fifos. Normally the shell script /dev/MAKEDEV is used to create special files for commonly known devices; it executes mknod with the appropriate arguments and can make all the files required for the device.

To make nodes manually, the arguments are:

-r Replace an existing file if its type is incorrect.

-R Replace an existing file if its type is incorrect. Correct the mode, user and group.

-g gid Specify the group for the device node. The gid operand may be a numeric group ID or a group name. If a group name is also a numeric group ID, the operand is used as a group name. Precede a numeric group ID with a # to stop it being treated as a name.

-m mode Specify the mode for the device node. The mode may be absolute or symbolic, see *chmod*.

-u uid Specify the user for the device node. The uid operand may be a numeric user ID or a user name. If a user name is also a numeric user ID, the operand is used as a user name. Precede a numeric user ID with a # to stop it being treated as a name.

name Device name, for example "tty" for a termios serial device or "hd" for a disk.

b | c | p Type of device. If the device is a block type device such as a tape or disk drive which needs both cooked and raw special files, the type is b. All other devices are character type devices, such as terminal and pseudo devices, and are type c. Specifying p creates fifo files.

driver | major The major device number is an integer number which tells the kernel which device driver entry point to use. If the device driver is configured into the current kernel it may be specified by driver name or major number.

minor The minor device number tells the kernel which one of several similar devices the node corresponds to; for example, it may be a specific serial port or pty.

unit and subunit The unit and subunit numbers select a subset of a device; for example, the unit may specify a particular disk, and the subunit a partition on that disk. (Currently this form of specification is only supported by the bsdos format, for compatibility with the BSD/OS mknod).

number A single opaque device number. Useful for netbooted computers which require device numbers packed in a format that isn't supported by -F.

EXIT STATUS:

The `mknod` utility exits 0 on success, and >0 if an error occurs.

NOTES:

NONE

EXAMPLES:

```
SHLL [/] mknod c 3 0 /dev/ttyS10
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MKNOD` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MKNOD` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mknod` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mknod(
  int    argc,
  char **argv
);
```

The configuration structure for the `mknod` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MKNOD_Command;
```

ORIGIN:

The implementation and portions of the documentation for this command are from NetBSD 4.0.

3.2.11 chroot - change the root directory

SYNOPSYS:

```
chroot [dir]
```

DESCRIPTION:

This command changes the root directory to `dir` for subsequent commands.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

The destination directory `dir` must exist.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `chroot` and the impact it has on the environment for subsequent command invocations:

```
SHLL [/] $ cat passwd
cat: passwd: No such file or directory
SHLL [/] $ chroot etc
SHLL [/] $ cat passwd
root:*:0:0:root::/:/bin/sh
rtems:*:1:1:RTEMS Application::/:/bin/sh
tty:!:2:2:tty owner::/:/bin/false
SHLL [/] $ cat /etc/passwd
cat: /etc/passwd: No such file or directory
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CHROOT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CHROOT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `chroot` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_chroot(
  int    argc,
  char **argv
);
```

The configuration structure for the `chroot` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CHROOT_Command;
```

3.2.12 chmod - change permissions of a file

SYNOPSYS:

```
chmod permissions file1 [file2...]
```

DESCRIPTION:

This command changes the permissions on the files specified to the indicated `permissions`. The permission values are POSIX based with owner, group, and world having individual read, write, and executive permission bits.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The `chmod` command only takes numeric representations of the permissions.

EXAMPLES:

The following is an example of how to use `chmod`:

```
SHLL [/] # cd etc
SHLL [/etc] # ls
-rw-r--r--   1   root   root         102 Jan 01 00:00 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:00 group
-rw-r--r--   1   root   root          30 Jan 01 00:00 issue
-rw-r--r--   1   root   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
SHLL [/etc] # chmod 0777 passwd
SHLL [/etc] # ls
-rwxrwxrwx   1   root   root         102 Jan 01 00:00 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:00 group
-rw-r--r--   1   root   root          30 Jan 01 00:00 issue
-rw-r--r--   1   root   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
SHLL [/etc] # chmod 0322 passwd
SHLL [/etc] # ls
--wx-w--w-   1 nouser   root         102 Jan 01 00:00 passwd
-rw-r--r--   1 nouser   root          42 Jan 01 00:00 group
-rw-r--r--   1 nouser   root          30 Jan 01 00:00 issue
-rw-r--r--   1 nouser   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
SHLL [/etc] # chmod 0644 passwd
SHLL [/etc] # ls
-rw-r--r--   1   root   root         102 Jan 01 00:00 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:00 group
-rw-r--r--   1   root   root          30 Jan 01 00:00 issue
-rw-r--r--   1   root   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CHMOD` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CHMOD` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `chmod` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_chmod(
  int    argc,
  char **argv
);
```

The configuration structure for the `chmod` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CHMOD_Command;
```

3.2.13 cat - display file contents

SYNOPSYS:

```
cat file1 [file2 .. fileN]
```

DESCRIPTION:

This command displays the contents of the specified files.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

It is possible to read the input from a device file using `cat`.

EXAMPLES:

The following is an example of how to use `cat`:

```
SHLL [/] # cat /etc/passwd
root:*:0:0:root::/:/bin/sh
rtems:*:1:1:RTEMS Application::/:/bin/sh
tty:!:2:2:tty owner::/:/bin/false
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CAT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CAT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `cat` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_cat(
  int    argc,
  char **argv
);
```

The configuration structure for the `cat` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CAT_Command;
```

3.2.14 rm - remove files

SYNOPSYS:

```
rm file1 [file2 ... fileN]
```

DESCRIPTION:

This command deletes a name from the filesystem. If the specified file name was the last link to a file and there are no `open` file descriptor references to that file, then it is deleted and the associated space in the file system is made available for subsequent use.

If the filename specified was the last link to a file but there are open file descriptor references to it, then the file will remain in existence until the last file descriptor referencing it is closed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `rm`:

```
SHLL [/] # cp /etc/passwd tmpfile
SHLL [/] # cat tmpfile
root:*:0:0:root::/:/bin/sh
rtems:*:1:1:RTEMS Application::/:/bin/sh
tty:!:2:2:tty owner::/:/bin/false
SHLL [/] # rm tmpfile
SHLL [/] # cat tmpfile
cat: tmpfile: No such file or directory
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_RM` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_RM` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `rm` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_rm(
  int    argc,
  char **argv
);
```

The configuration structure for the `rm` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_RM_Command;
```

3.2.15 mount - mount disk

SYNOPSYS:

```
mount [-t fstype] [-r] [-L] device path
```

DESCRIPTION:

The `mount` command will mount a block device to a mount point using the specified file system. The files systems are:

- msdos - MSDOS File System
- tftp - TFTP Network File System
- ftp - FTP Network File System
- nfs - Network File System
- rfs - RTEMS File System

When the file system type is 'msdos' or 'rfs' the driver is a `"block device driver"` node present in the file system. The driver is ignored with the 'tftp' and 'ftp' file systems. For the 'nfs' file system the driver is the 'host:/path' string that described NFS host and the exported file system path.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The mount point must exist.

The services offered by each file-system vary. For example you cannot list the directory of a TFTP file-system as this server is not provided in the TFTP protocol. You need to check each file-system's documentation for the services provided.

EXAMPLES:

Mount the Flash Disk driver to the '/fd' mount point:

```
SHLL [/] $ mount -t msdos /dev/flashdisk0 /fd
```

Mount the NFS file system exported path 'bar' by host 'foo':

```
$ mount -t nfs foo:/bar /nfs
```

Mount the TFTP file system on '/tftp':

```
$ mount -t tftp /tftp
```

To access the TFTP files on server '10.10.10.10':

```
$ cat /tftp/10.10.10.10/test.txt
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MOUNT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MOUNT` when all shell commands have been configured.

The mount command includes references to file-system code. If you do not wish to include file-system that you do not use do not define the mount command support for that file-system. The file-system mount command defines are:

- msdos - CONFIGURE_SHELL_MOUNT_MSDOS
- tftp - CONFIGURE_SHELL_MOUNT_TFTP
- ftp - CONFIGURE_SHELL_MOUNT_FTP
- nfs - CONFIGURE_SHELL_MOUNT_NFS
- rfs - CONFIGURE_SHELL_MOUNT_RFS

An example configuration is:

```
#define CONFIGURE_SHELL_MOUNT_MSDOS
#ifdef RTEMS_NETWORKING
  #define CONFIGURE_SHELL_MOUNT_TFTP
  #define CONFIGURE_SHELL_MOUNT_FTP
  #define CONFIGURE_SHELL_MOUNT_NFS
  #define CONFIGURE_SHELL_MOUNT_RFS
#endif
```

PROGRAMMING INFORMATION:

The `mount` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mount(
  int    argc,
  char **argv
);
```

The configuration structure for the `mount` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MOUNT_Command;
```

3.2.16 unmount - unmount disk

SYNOPSYS:

```
unmount path
```

DESCRIPTION:

This command unmounts the device at the specified `path`.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

TBD - Surely there must be some warnings to go here.

EXAMPLES:

The following is an example of how to use `unmount`:

```
EXAMPLE_TBD
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_UNMOUNT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_UNMOUNT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `unmount` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_unmount(
  int    argc,
  char **argv
);
```

The configuration structure for the `unmount` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_UNMOUNT_Command;
```

3.2.17 blkssync - sync the block driver

SYNOPSYS:

```
blksync driver
```

DESCRIPTION:

This command XXX

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `blksync`:

```
EXAMPLE_TBD
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_BLKSYNC` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_BLKSYNC` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `blksync` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_blksync(
  int    argc,
  char **argv
);
```

The configuration structure for the `blksync` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_BLKSYNC_Command;
```

3.2.18 dd - convert and copy a file

SYNOPSYS:

```
dd [operands ...]
```

DESCRIPTION:

The dd utility copies the standard input to the standard output. Input data is read and written in 512-byte blocks. If input reads are short, input from multiple reads are aggregated to form the output block. When finished, dd displays the number of complete and partial input and output blocks and truncated input records to the standard error output.

The following operands are available:

bs=n Set both input and output block size, superseding the ibs and obs operands. If no conversion values other than noerror, notrunc or sync are specified, then each input block is copied to the output as a single block without any aggregation of short blocks.

cbs=n Set the conversion record size to n bytes. The conversion record size is required by the record oriented conversion values.

count=n Copy only n input blocks.

files=n Copy n input files before terminating. This operand is only applicable when the input device is a tape.

ibs=n Set the input block size to n bytes instead of the default 512.

if=file Read input from file instead of the standard input.

obs=n Set the output block size to n bytes instead of the default 512.

of=file Write output to file instead of the standard output. Any regular output file is truncated unless the notrunc conversion value is specified. If an initial portion of the output file is skipped (see the seek operand) the output file is truncated at that point.

seek=n Seek n blocks from the beginning of the output before copying. On non-tape devices, a *lseek* operation is used. Otherwise, existing blocks are read and the data discarded. If the seek operation is past the end of file, space from the current end of file to the specified offset is filled with blocks of NUL bytes.

skip=n Skip n blocks from the beginning of the input before copying. On input which supports seeks, a *lseek* operation is used. Otherwise, input data is read and discarded. For pipes, the correct number of bytes is read. For all other devices, the correct number of blocks is read without distinguishing between a partial or complete block being read.

progress=n Switch on display of progress if n is set to any non-zero value. This will cause a "." to be printed (to the standard error output) for every n full or partial blocks written to the output file.

conv=value[,value...] Where value is one of the symbols from the following list.

ascii, oldascii The same as the unblock value except that characters are translated from EBCDIC to ASCII before the records are converted. (These values imply unblock if the operand cbs is also specified.) There are two conversion maps for ASCII. The value ascii specifies the recom- mended one which is compatible with AT&T System V UNIX. The value oldascii specifies the one used in historic AT&T and pre 4.3BSD-Reno systems.

block Treats the input as a sequence of newline or end-of-file terminated variable length records independent of input and output block boundaries. Any trailing newline character is discarded. Each input record is converted to a fixed length output record where the length is specified by the cbs operand. Input records shorter than the conversion record size are padded with spaces. Input records longer than the conversion record size are truncated. The number of truncated input records, if any, are reported to the standard error output at the completion of the copy.

ebcdic, ibm, oldebcdic, oldibm
The same as the block value except that characters are translated from ASCII to EBCDIC after the records are converted. (These values imply block if the operand cbs is also specified.) There are four conversion maps for EBCDIC. The value ebcdic specifies the recommended one which is compatible with AT&T System V UNIX. The value ibm is a slightly different mapping, which is compatible with the AT&T System V UNIX ibm value. The values oldebcdic and oldibm are maps used in historic AT&T and pre 4.3BSD-Reno systems.

lcase Transform uppercase characters into lowercase characters.

noerror Do not stop processing on an input error. When an input error occurs, a diagnostic message followed by the current input and output block counts will be written to the standard error output in the same format as the standard completion message. If the sync conversion is also specified, any missing input data will be replaced with NUL bytes (or with spaces if a block oriented conversion value was specified) and processed as a normal input buffer. If the sync conversion is not specified, the input block is omitted from the output. On input files which are not tapes or pipes, the file offset will be positioned past the block in which the error occurred using lseek(2).

notrunc Do not truncate the output file. This will preserve any blocks in the output file not explicitly written by dd. The notrunc value is not supported for tapes.

osync Pad the final output block to the full output block size. If the input file is not a multiple of the output block size after conversion, this conversion forces the final output block to be the same size as preceding blocks for use on devices that require regularly sized blocks to be written. This option is incompatible with use of the bs=n block size specification.

sparse If one or more non-final output blocks would consist solely of NUL bytes, try to seek the output file by the required space instead of filling them with NULs. This results in a sparse file on some file systems.

swab Swap every pair of input bytes. If an input buffer has an odd number of bytes, the last byte will be ignored during swapping.

sync Pad every input block to the input buffer size. Spaces are used for pad bytes if a block oriented conversion value is specified, otherwise NUL bytes are used.

ucase Transform lowercase characters into uppercase characters.

unblock Treats the input as a sequence of fixed length records independent of input and output block boundaries. The length of the input records is specified by the cbs operand. Any trailing space characters are discarded and a newline character is appended.

Where sizes are specified, a decimal number of bytes is expected. Two or more numbers may be separated by an "x" to indicate a product. Each number may have one of the following optional suffixes:

b Block; multiply by 512

k Kibi; multiply by 1024 (1 KiB)

m Mebi; multiply by 1048576 (1 MiB)

g Gibi; multiply by 1073741824 (1 GiB)

t Tebi; multiply by 1099511627776 (1 TiB)

w Word; multiply by the number of bytes in an integer

When finished, dd displays the number of complete and partial input and output blocks, truncated input records and odd-length byte-swapping ritten. Partial output blocks to tape devices are considered fatal errors. Otherwise, the rest of the block will be written. Partial output blocks to character devices will produce a warning message. A truncated input block is one where a variable length record oriented conversion value was specified and the input line was too long to fit in the conversion record or was not newline terminated.

Normally, data resulting from input or conversion or both are aggregated into output blocks of the specified size. After the end of input is reached, any remaining output is written as a block. This means that the final output block may be shorter than the output block size.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `dd`:

```
SHLL [/] $ dd if=/nfs/boot-image of=/dev/hda1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DD` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DD` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `dd` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_dd(
  int    argc,
  char **argv
```

```
);
```

The configuration structure for the `dd` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DD_Command;
```

3.2.19 hexdump - ascii/dec/hex/octal dump

SYNOPSYS:

```
hexdump [-bcCdovx] [-e format_string] [-f format_file] [-n length]
        [-s skip] file ...
```

DESCRIPTION:

The hexdump utility is a filter which displays the specified files, or the standard input, if no files are specified, in a user specified format.

The options are as follows:

-b One-byte octal display. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, zero-filled, bytes of input data, in octal, per line.

-c One-byte character display. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, space-filled, characters of input data per line.

-C Canonical hex+ASCII display. Display the input offset in hexadecimal, followed by sixteen space-separated, two column, hexadecimal bytes, followed by the same sixteen bytes in %_p format enclosed in “|” characters.

-d Two-byte decimal display. Display the input offset in hexadecimal, followed by eight space-separated, five column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

-e format_string Specify a format string to be used for displaying data.

-f format_file Specify a file that contains one or more newline separated format strings. Empty lines and lines whose first non-blank character is a hash mark (#) are ignored.

-n length Interpret only length bytes of input.

-o Two-byte octal display. Display the input offset in hexadecimal, followed by eight space-separated, six column, zerofilled, two byte quantities of input data, in octal, per line.

-s offset Skip offset bytes from the beginning of the input. By default, offset is interpreted as a decimal number. With a leading 0x or 0X, offset is interpreted as a hexadecimal number, otherwise, with a leading 0, offset is interpreted as an octal number. Appending the character b, k, or m to offset causes it to be interpreted as a multiple of 512, 1024, or 1048576, respectively.

-v The -v option causes hexdump to display all input data. Without the -v option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line containing a single asterisk.

-x Two-byte hexadecimal display. Display the input offset in hexadecimal, followed by eight, space separated, four column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.

For each input file, hexdump sequentially copies the input to standard output, transforming the data according to the format strings specified by the -e and -f options, in the order that they were specified.

Formats

A format string contains any number of format units, separated by whitespace. A format unit contains up to three items: an iteration count, a byte count, and a format.

The iteration count is an optional positive integer, which defaults to one. Each format is applied iteration count times.

The byte count is an optional positive integer. If specified it defines the number of bytes to be interpreted by each iteration of the format.

If an iteration count and/or a byte count is specified, a single slash must be placed after the iteration count and/or before the byte count to disambiguate them. Any whitespace before or after the slash is ignored.

The format is required and must be surrounded by double quote (" ") marks. It is interpreted as a fprintf-style format string (see *fprintf*), with the following exceptions:

- An asterisk (*) may not be used as a field width or precision.
- A byte count or field precision is required for each "s" con- version character (unlike the fprintf(3) default which prints the entire string if the precision is unspecified).
- The conversion characters "h", "l", "n", "p" and "q" are not supported.
- The single character escape sequences described in the C standard are supported:

  NUL \0 <alert character> \a <backspace> \b <form-feed> \f <newline> \n <carriage return> \r <tab> \t <vertical tab> \v

Hexdump also supports the following additional conversion strings:

_a[dox] Display the input offset, cumulative across input files, of the next byte to be displayed. The appended characters d, o, and x specify the display base as decimal, octal or hexadecimal respectively.

_A[dox] Identical to the _a conversion string except that it is only performed once, when all of the input data has been processed.

_c Output characters in the default character set. Nonprinting characters are displayed in three character, zero-padded octal, except for those representable by standard escape notation (see above), which are displayed as two character strings.

_p Output characters in the default character set. Nonprinting characters are displayed as a single ".".

_u Output US ASCII characters, with the exception that control characters are displayed using the following, lower-case, names. Characters greater than 0xff, hexadecimal, are displayed as hexadecimal strings.

000 nul 001 soh 002 stx 003 etx 004 eot 005 enq 006 ack 007 bel 008 bs 009 ht 00A lf 00B vt 00C ff 00D cr 00E so 00F si 010 dle 011 dc1 012 dc2 013 dc3 014 dc4 015 nak 016 syn 017 etb 018 can 019 em 01A sub 01B esc 01C fs 01D gs 01E rs 01F us 07F del

The default and supported byte counts for the conversion characters are as follows:

%_c, %_p, %_u, %c One byte counts only.

%d, %i, %o, %u, %X, %x Four byte default, one, two, four and eight byte counts supported.

%E, %e, %f, %G, %g Eight byte default, four byte counts supported.

The amount of data interpreted by each format string is the sum of the data required by each format unit, which is the iteration count times the byte count, or the iteration count times the number of bytes required by the format if the byte count is not specified.

The input is manipulated in "blocks", where a block is defined as the largest amount of data specified by any format string. Format strings interpreting less than an input block's worth of data, whose last format unit both interprets some number of bytes and does not have a specified iteration count, have the iteration count incremented until the entire input block has been processed or there is not enough data remaining in the block to satisfy the format string.

If, either as a result of user specification or hexdump modifying the iteration count as described above, an iteration count is greater than one, no trailing whitespace characters are output during the last iteration.

It is an error to specify a byte count as well as multiple conversion characters or strings unless all but one of the conversion characters or strings is _a or _A.

If, as a result of the specification of the -n option or end-of-file being reached, input data only partially satisfies a format string, the input block is zero-padded sufficiently to display all available data (i.e. any format units overlapping the end of data will display some number of the zero bytes).

Further output by such format strings is replaced by an equivalent number of spaces. An equivalent number of spaces is defined as the number of spaces output by an s conversion character with the same field width and precision as the original conversion character or conversion string but with any "+", " ", "#" conversion flag characters removed, and referencing a NULL string.

If no format strings are specified, the default display is equivalent to specifying the -x option.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `hexdump`:

```
SHLL [/] $ hexdump -C -n 512 /dev/hda1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_HEXDUMP` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_HEXDUMP` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `hexdump` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_hexdump(
  int    argc,
  char **argv
);
```

The configuration structure for the `hexdump` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_HEXDUMP_Command;
```

3.2.20 fdisk - format disk

SYNOPSYS:

```
fdisk
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_FDISK` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_FDISK` when all shell commands have been configured.

3.2.21 dir - alias for ls

SYNOPSYS:

```
dir [dir]
```

DESCRIPTION:

This command is an alias or alternate name for the `ls`. See Section 3.2.5 [File and Directory Commands ls - list files in the directory], page 31 for more information.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `dir`:

```
SHLL [/] $ dir
drwxr-xr-x   1   root   root         536 Jan 01 00:00 dev/
drwxr-xr-x   1   root   root        1072 Jan 01 00:00 etc/
2 files 1608 bytes occupied
SHLL [/] $ dir etc
-rw-r--r--   1   root   root         102 Jan 01 00:00 passwd
-rw-r--r--   1   root   root          42 Jan 01 00:00 group
-rw-r--r--   1   root   root          30 Jan 01 00:00 issue
-rw-r--r--   1   root   root          28 Jan 01 00:00 issue.net
4 files 202 bytes occupied
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DIR` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DIR` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `dir` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_dir(
  int    argc,
  char **argv
);
```

The configuration structure for the `dir` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DIR_Command;
```

3.2.22 mkrfs - format RFS file system

SYNOPSYS:

```
mkrfs [-vsbiIo] device
```

DESCRIPTION:

Format the block device with the RTEMS File System (RFS). The default configuration with not parameters selects a suitable block size based on the size of the media being formatted.

The media is broken up into groups of blocks. The number of blocks in a group is based on the number of bits a block contains. The large a block the more blocks a group contains and the fewer groups in the file system.

The following options are provided:

-v Display configuration and progress of the format.

-s Set the block size in bytes.

-b The number of blocks in a group. The block count must be equal or less than the number of bits in a block.

-i Number of inodes in a group. The inode count must be equal or less than the number of bits in a block.

-I Initialise the inodes. The default is not to initialise the inodes and to rely on the inode being initialised when allocated. Initialising the inode table helps recovery if a problem appears.

-o Integer percentage of the media used by inodes. The default is 1%.

device Path of the device to format.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `mkrfs`:

```
SHLL [/] $ mkrfs /dev/fdda
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MKRFS` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MKRFS` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mkrfs` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mkrfs(
  int    argc,
  char **argv
);
```

The configuration structure for `mkrfs` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MKRFS_Command;
```

3.2.23 debugrfs - debug RFS file system

SYNOPSYS:

```
debugrfs [-hl] path command [options]
```

DESCRIPTION:

The command provides debugging information for the RFS file system.

The options are:

-h Print a help message.

-l List the commands.

path Path to the mounted RFS file system. The file system has to be mounted to view to use this command.

The commands are:

block start [end] Display the contents of the blocks from start to end.

data Display the file system data and configuration.

dir bno Process the block as a directory displaying the entries.

group start [end] Display the group data from the start group to the end group.

inode [-aef] [start] [end]
Display the inodes between start and end. If no start and end is provides all inodes are displayed.

- **-a** Display all inodes. That is allocated and unallocated inodes.
- **-e** Search and display on inodes that have an error.
- **-f** Force display of inodes, even when in error.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `debugrfs`:

```
SHLL [/] $ debugrfs /c data
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DEBUGRFS` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DEBUGRFS` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `debugrfs` command is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_debugrfs(
  int    argc,
  char **argv
);
```

The configuration structure for `debugrfs` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DEBUGRFS_Command;
```

3.2.24 cd - alias for chdir

SYNOPSYS:

```
cd directory
```

DESCRIPTION:

This command is an alias or alternate name for the `chdir`. See Section 3.2.6 [File and Directory Commands chdir - change the current directory], page 33 for more information.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `cd`:

```
SHLL [/] $ cd etc
SHLL [/etc] $ cd /
SHLL [/] $ cd /etc
SHLL [/etc] $ pwd
/etc
SHLL [/etc] $ cd /
SHLL [/] $ pwd
/
SHLL [/] $ cd etc
SHLL [/etc] $ cd ..
SHLL [/] $ pwd
/
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CD` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CD` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `cd` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_cd(
  int    argc,
  char **argv
);
```

The configuration structure for the `cd` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CD_Command;
```

4 Memory Commands

4.1 Introduction

The RTEMS shell has the following memory commands:

- `mdump` - Display contents of memory
- `wdump` - Display contents of memory (word)
- `medit` - Modify contents of memory
- `mfill` - File memory with pattern
- `mmove` - Move contents of memory
- `malloc` - Obtain information on C Program Heap

4.2 Commands

This section details the Memory Commands available. A subsection is dedicated to each of the commands and describes the behavior and configuration of that command as well as providing an example usage.

4.2.1 mdump - display contents of memory

SYNOPSYS:

```
mdump [address [length]]
```

DESCRIPTION:

This command displays the contents of memory at the `address` and `length` in bytes specified on the command line.

When `length` is not provided, it defaults to `320` which is twenty lines of output with sixteen bytes of output per line.

When `address` is not provided, it defaults to `0x00000000`.

EXIT STATUS:

This command always returns 0 to indicate success.

NOTES:

Dumping memory from a non-existent address may result in an unrecoverable program fault.

EXAMPLES:

The following is an example of how to use `mdump`:

```
SHLL [/] $ mdump 0x10000 32
0x0001000000 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0x0001001000 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
SHLL [/] $ mdump 0x02000000 32
0x02000000A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)..3.."...!.
0x02000010A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 01 .H..)..3.."...!.
SHLL [/] $ mdump 0x02001000 32
0x0200100003 00 80 00 82 10 60 00-81 98 40 00 83 48 00 00 ......`.....H..
0x0200101084 00 60 01 84 08 A0 07-86 10 20 01 87 28 C0 02 ..`....... ..(..
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MDUMP` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MDUMP` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mdump` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mdump(
  int    argc,
  char **argv
);
```

The configuration structure for the `mdump` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MDUMP_Command;
```

4.2.2 wdump - display contents of memory (word)

SYNOPSYS:

```
wdump [address [length]]
```

DESCRIPTION:

This command displays the contents of memory at the `address` and `length` in bytes specified on the command line.

When `length` is not provided, it defaults to `320` which is twenty lines of output with sixteen bytes of output per line.

When `address` is not provided, it defaults to `0x00000000`.

EXIT STATUS:

This command always returns 0 to indicate success.

NOTES:

Dumping memory from a non-existent address may result in an unrecoverable program fault.

EXAMPLES:

The following is an example of how to use `wdump`:

```
SHLL [/] $ wdump 0x02010000 32
0x02010000 0201 08D8 0201 08C0-0201 08AC 0201 0874 ...............t
0x02010010 0201 0894 0201 0718-0201 0640 0201 0798 ...............
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_WDUMP` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_WDUMP` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `wdump` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_wdump(
  int    argc,
  char **argv
);
```

The configuration structure for the `wdump` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_WDUMP_Command;
```

4.2.3 medit - modify contents of memory

SYNOPSYS:

```
medit address value1 [value2 ... valueN]
```

DESCRIPTION:

This command is used to modify the contents of the memory starting at `address` using the octets specified by the parameters `value1` through `valueN`.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

Dumping memory from a non-existent address may result in an unrecoverable program fault.

EXAMPLES:

The following is an example of how to use `medit`:

```
SHLL [/] $ mdump 0x02000000 32
0x02000000 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)..3.."...!.
0x02000010 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 01 .H..)..3.."...!.
SHLL [/] $  medit 0x02000000 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
SHLL [/] $ mdump 0x02000000 32
0x02000000 01 02 03 04 05 06 07 08-09 00 22 BC A6 10 21 00 .........."...!.
0x02000010 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 01 .H..)..3.."...!.
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MEDIT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MEDIT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `medit` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_medit(
  int    argc,
  char **argv
);
```

The configuration structure for the `medit` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MEDIT_Command;
```

4.2.4 mfill - file memory with pattern

SYNOPSYS:

```
mfill address length value
```

DESCRIPTION:

This command is used to fill the memory starting at `address` for the specified `length` in octets when the specified at `value`.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

Filling a non-existent address range may result in an unrecoverable program fault. Similarly overwriting interrupt vector tables, code space or critical data areas can be fatal as shown in the example.

EXAMPLES:

In this example, the address used (`0x23d89a0`) as the base address of the filled area is the end of the stack for the Idle thread. This address was determined manually using gdb and is very specific to this application and BSP. The first command in this example is an `mdump` to display the initial contents of this memory. We see that the first 8 bytes are 0xA5 which is the pattern used as a guard by the Stack Checker. On the first context switch after the pattern is overwritten by the `mfill` command, the Stack Checker detect the pattern has been corrupted and generates a fatal error.

```
SHLL [/] $ mdump 0x23d89a0 16
0x023D89A0 A5 A5 A5 A5 A5 A5 A5 A5-FE ED F0 0D 0B AD 0D 06 ................
SHLL [/] $ mfill 0x23d89a0 13 0x5a
SHLL [/] $ BLOWN STACK!!! Offending task(0x23D4418): id=0x09010001; name=0x0203D908
  stack covers range 0x23D89A0 - 0x23D99AF (4112 bytes)
  Damaged pattern begins at 0x023D89A8 and is 16 bytes long
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MFILL` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MFILL` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mfill` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mfill(
  int    argc,
  char **argv
);
```

The configuration structure for the `mfill` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MFILL_Command;
```

4.2.5 mmove - move contents of memory

SYNOPSYS:

```
mmove dst src length
```

DESCRIPTION:

This command is used to copy the contents of the memory starting at `src` to the memory located at `dst` for the specified `length` in octets.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `mmove`:

```
SHLL [/] $ mdump 0x023d99a0 16
0x023D99A0 A5 A5 A5 A5 A5 A5 A5 A5-A5 A5 A5 A5 A5 A5 A5 A5 ................
SHLL [/] $ mdump 0x02000000 16
0x02000000 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)..3.."...!.
SHLL [/] $ mmove 0x023d99a0 0x02000000 13
SHLL [/] $ mdump 0x023d99a0 16
0x023D99A0 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 A5 A5 A5 .H..)..3..".....
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MMOVE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MMOVE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `mmove` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_mmove(
  int    argc,
  char **argv
);
```

The configuration structure for the `mmove` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MMOVE_Command;
```

4.2.6 malloc - obtain information on C program heap

SYNOPSYS:

```
malloc [info|stats]
```

DESCRIPTION:

This command prints either information or statistics about the C Program Heap used by the `malloc` family of calls based upon the value of the first argument passed to the command.

When the subcommand `info` is specified, information on the current state of the C Program Heap is reported. This includes the following information:

- Number of free blocks
- Largest free block
- Total bytes free
- Number of used blocks
- Largest used block
- Total bytes used

When the subcommand `stats` is specified, statistics on the the C Program Heap are reported. Malloc Family Statistics must be enabled for all of the values to be updated. The statistics available includes the following information:

-
- Currently available memory (in kilobytes)
- Currently allocated memory (in kilobytes)
- Maximum amount of memory ever allocated (in kilobytes)
- Lifetime tally of allocated memory (in kilobytes)
- Lifetime tally of freed memory (in kilobytes)
- Number of calls to `malloc`
- Number of calls to `free`
- Number of calls to `realloc`
- Number of calls to `calloc`

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The `CONFIGURE_MALLOC_STATISTICS` `confdefs.h` constant must be defined when the application is configured for the full set of statistics information to be available.

EXAMPLES:

The following is an example of how to use the `malloc` command.

```
SHLL [/] $ malloc info
Number of free blocks: 3
Largest free block:    3626672
Total bytes free:      3627768
Number of used blocks: 130
Largest used block:    1048
Total bytes used:      10136
SHLL [/] $ malloc stats
Malloc statistics
  avail:3552k  allocated:9k (0%) max:10k (0%) lifetime:21k freed:12k
  Call counts:   malloc:203   free:93   realloc:0   calloc:20
SHLL [/] $ malloc info
Number of free blocks: 3
Largest free block:    3626672
Total bytes free:      3627768
Number of used blocks: 130
Largest used block:    1048
Total bytes used:      10136
SHLL [/] $ malloc stats
Malloc statistics
  avail:3552k  allocated:9k (0%) max:10k (0%) lifetime:23k freed:14k
  Call counts:   malloc:205   free:95   realloc:0   calloc:20
```

Note that in the above example, the lifetime allocated and free values have increased between the two calls to `malloc stats` even though the amount of memory available in the C Program Heap is the same in both the `malloc info` invocations. This indicates that memory was allocated and freed as a side-effect of the commands.

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_MALLOC` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_MALLOC` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `malloc` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_malloc(
  int    argc,
  char **argv
);
```

The configuration structure for the `malloc` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_MALLOC_Command;
```

5 RTEMS Specific Commands

5.1 Introduction

The RTEMS shell has the following rtems commands:

- `halt` - Shutdown the system
- `cpuuse` - print or reset per thread cpu usage
- `stackuse` - print per thread stack usage
- `perioduse` - print or reset per period usage
- `wkspace` - Display information on Executive Workspace
- `config` - Show the system configuration.
- `itask` - List init tasks for the system
- `extension` - Display information about extensions
- `task` - Display information about tasks
- `queue` - Display information about message queues
- `sema` - display information about semaphores
- `region` - display information about regions
- `part` - display information about partitions
- `object` - Display information about RTEMS objects
- `driver` - Display the RTEMS device driver table
- `dname` - Displays information about named drivers
- `pthread` - Displays information about POSIX threads

5.2 Commands

This section details the RTEMS Specific Commands available. A subsection is dedicated to each of the commands and describes the behavior and configuration of that command as well as providing an example usage.

5.2.1 halt - Shutdown the system

SYNOPSYS:

```
halt
```

DESCRIPTION:

This command is used to shutdown the RTEMS application.

EXIT STATUS:

This command does not return.

NOTES:

EXAMPLES:

The following is an example of how to use `halt`:

```
SHLL [/] $ halt
```

The user will not see another prompt and the system will shutdown.

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_HALT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_HALT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `halt` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_halt(
  int    argc,
  char **argv
);
```

The configuration structure for the `halt` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_HALT_Command;
```

5.2.2 cpuuse - print or reset per thread cpu usage

SYNOPSYS:

```
cpuuse [-r]
```

DESCRIPTION:

This command may be used to print a report on the per thread cpu usage or to reset the per thread CPU usage statistics. When invoked with the `-r` option, the CPU usage statistics are reset.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The granularity of the timing information reported is dependent upon the BSP and the manner in which RTEMS was built. In the default RTEMS configuration, if the BSP supports nanosecond granularity timestamps, then the information reported will be highly accurate. Otherwise, the accuracy of the information reported is limited by the clock tick quantum.

EXAMPLES:

The following is an example of how to use `cpuuse`:

```
SHLL [/] $ cpuuse
CPU Usage by thread
   ID            NAME         SECONDS   PERCENT
0x09010001   IDLE           49.745393   98.953
0x0a010001   UI1             0.000000    0.000
0x0a010002   SHLL            0.525928    1.046
Time since last CPU Usage reset 50.271321 seconds
SHLL [/] $ cpuuse -r
Resetting CPU Usage information
SHLL [/] $ cpuuse
CPU Usage by thread
   ID            NAME         SECONDS   PERCENT
0x09010001   IDLE            0.000000    0.000
0x0a010001   UI1             0.000000    0.000
0x0a010002   SHLL            0.003092  100.000
Time since last CPU Usage reset 0.003092 seconds
```

In the above example, the system had set idle for nearly a minute when the first report was generated. The `cpuuse -r` and `cpuuse` commands were pasted from another window so were executed with no gap between. In the second report, only the `shell` thread has run since the CPU Usage was reset. It has consumed approximately 3.092 milliseconds of CPU time processing the two commands and generating the output.

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CPUUSE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CPUUSE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `cpuuse` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_cpuuse(
  int    argc,
  char **argv
);
```

The configuration structure for the `cpuuse` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CPUUSE_Command;
```

5.2.3 stackuse - print per thread stack usage

SYNOPSYS:

```
stackuse
```

DESCRIPTION:

This command prints a Stack Usage Report for all of the tasks and threads in the system. On systems which support it, the usage of the interrupt stack is also included in the report.

EXIT STATUS:

This command always succeeds and returns 0.

NOTES:

The `CONFIGURE_STACK_CHECKER_ENABLED` `confdefs.h` constant must be defined when the application is configured for this command to have any information to report.

EXAMPLES:

The following is an example of how to use `stackuse`:

```
SHLL [/] $ stackuse
Stack usage by thread
    ID      NAME    LOW          HIGH     CURRENT     AVAILABLE     USED
0x09010001  IDLE 0x023d89a0 - 0x023d99af 0x023d9760      4096        608
0x0a010001  UI1  0x023d9f30 - 0x023daf3f 0x023dad18      4096       1804
0x0a010002  SHLL 0x023db4c0 - 0x023df4cf 0x023de9d0     16384       5116
0xffffffff  INTR 0x023d2760 - 0x023d375f 0x00000000      4080        316
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_STACKUSE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_STACKUSE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `stackuse` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_stackuse(
  int    argc,
  char **argv
);
```

The configuration structure for the `stackuse` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_STACKUSE_Command;
```

5.2.4 perioduse - print or reset per period usage

SYNOPSYS:

```
perioduse [-r]
```

DESCRIPTION:

This command may be used to print a statistics report on the rate monotonic periods in the application or to reset the rate monotonic period usage statistics. When invoked with the `-r` option, the usage statistics are reset.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

The granularity of the timing information reported is dependent upon the BSP and the manner in which RTEMS was built. In the default RTEMS configuration, if the BSP supports nanosecond granularity timestamps, then the information reported will be highly accurate. Otherwise, the accuracy of the information reported is limited by the clock tick quantum.

EXAMPLES:

The following is an example of how to use `perioduse`:

```
SHLL [/] $ perioduse
Period information by period
--- CPU times are in seconds ---
--- Wall times are in seconds ---
   ID     OWNER COUNT MISSED          CPU TIME                  WALL TIME
                                    MIN/MAX/AVG              MIN/MAX/AVG
0x42010001 TA1    502      0 0:000039/0:042650/0:004158 0:000039/0:020118/0:002848
0x42010002 TA2    502      0 0:000041/0:042657/0:004309 0:000041/0:020116/0:002848
0x42010003 TA3    501      0 0:000041/0:041564/0:003653 0:000041/0:020003/0:002814
0x42010004 TA4    501      0 0:000043/0:044075/0:004911 0:000043/0:020004/0:002814
0x42010005 TA5     10      0 0:000065/0:005413/0:002739 0:000065/1:000457/0:041058

                                    MIN/MAX/AVG              MIN/MAX/AVG
SHLL [/] $ perioduse -r
Resetting Period Usage information
SHLL [/] $ perioduse
--- CPU times are in seconds ---
--- Wall times are in seconds ---
   ID     OWNER COUNT MISSED          CPU TIME                  WALL TIME
                                    MIN/MAX/AVG              MIN/MAX/AVG
0x42010001 TA1      0      0
0x42010002 TA2      0      0
0x42010003 TA3      0      0
0x42010004 TA4      0      0
0x42010005 TA5      0      0
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_PERIODUSE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_PERIODUSE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `perioduse` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_perioduse(
  int    argc,
  char **argv
);
```

The configuration structure for the `perioduse` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_PERIODUSE_Command;
```

5.2.5 wkspace - display information on executive workspace

SYNOPSYS:

```
wkspace
```

DESCRIPTION:

This command prints information on the current state of the RTEMS Executive Workspace reported. This includes the following information:

- Number of free blocks
- Largest free block
- Total bytes free
- Number of used blocks
- Largest used block
- Total bytes used

EXIT STATUS:

This command always succeeds and returns 0.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `wkspace`:

```
SHLL [/] $ wkspace
Number of free blocks: 1
Largest free block:    132336
Total bytes free:      132336
Number of used blocks: 36
Largest used block:    16408
Total bytes used:      55344
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_WKSPACE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_WKSPACE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `wkspace` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_wkspace(
  int    argc,
  char **argv
);
```

The configuration structure for the `wkspace` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_WKSPACE_Command;
```

5.2.6 config - show the system configuration.

SYNOPSYS:

```
config
```

DESCRIPTION:

This command display information about the RTEMS Configuration.

EXIT STATUS:

This command always succeeds and returns 0.

NOTES:

At this time, it does not report every configuration parameter. This is an area in which user submissions or sponsorship of a developer would be appreciated.

EXAMPLES:

The following is an example of how to use `config`:

```
INITIAL (startup) Configuration Info
------------------------------------------------------------------------------
WORKSPACE      start: 0x23d22e0;  size: 0x2dd20
TIME           usec/tick: 10000;  tick/timeslice: 50;  tick/sec: 100
MAXIMUMS       tasks: 20;  timers: 0;  sems: 50;  que's: 20;  ext's: 1
               partitions: 0;  regions: 0;  ports: 0;  periods: 0
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_CONFIG` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_CONFIG` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `config` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_config(
  int    argc,
  char **argv
);
```

The configuration structure for the `config` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_CONFIG_Command;
```

5.2.7 itask - list init tasks for the system

SYNOPSYS:

```
itask
```

DESCRIPTION:

This command prints a report on the set of initialization tasks and threads in the system.

EXIT STATUS:

This command always succeeds and returns 0.

NOTES:

At this time, it includes only Classic API Initialization Tasks. This is an area in which user submissions or sponsorship of a developer would be appreciated.

EXAMPLES:

The following is an example of how to use `itask`:

```
SHLL [/] $ itask
  #    NAME   ENTRY        ARGUMENT    PRIO   MODES  ATTRIBUTES   STACK SIZE
------------------------------------------------------------------------------
   0   UI1    [0x2002258] 0 [0x0]         1    nP      DEFAULT     4096 [0x1000]
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_ITASK` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_ITASK` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `itask` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_itask(
  int    argc,
  char **argv
);
```

The configuration structure for the `itask` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_ITASK_Command;
```

5.2.8 extension - display information about extensions

SYNOPSYS:

```
extension [id [id ...] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of User Extensions currently active in the system.

If invoked with a set of ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of using the `extension` command on a system with no user extensions.

```
SHLL [/] $ extension
  ID       NAME
------------------------------------------------------------------------------
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_EXTENSION` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_EXTENSION` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `extension` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_extension(
  int    argc,
  char **argv
);
```

The configuration structure for the `extension` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_EXTENSION_Command;
```

5.2.9 task - display information about tasks

SYNOPSYS:

```
task [id [id ...] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Classic API Tasks currently active in the system.

If invoked with a set of ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use the `task` on an application with just two Classic API tasks:

```
SHLL [/] $ task
  ID       NAME   PRIO   STAT   MODES  EVENTS   WAITID  WAITARG  NOTES
------------------------------------------------------------------------------
0a010001   UI1      1   SUSP   P:T:nA  NONE
0a010002   SHLL   100   READY  P:T:nA  NONE
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_TASK` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_TASK` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `task` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_task(
  int    argc,
  char **argv
);
```

The configuration structure for the `task` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_TASK_Command;
```

5.2.10 queue - display information about message queues

SYNOPSYS:

```
queue [id [id ... ] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Classic API Message Queues currently active in the system.

If invoked with a set of ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of using the `queue` command on a system with no Classic API Message Queues.

```
SHLL [/] $ queue
  ID       NAME   ATTRIBUTES   PEND   MAXPEND  MAXSIZE
------------------------------------------------------------------------------
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_QUEUE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_QUEUE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `queue` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_queue(
  int    argc,
  char **argv
);
```

The configuration structure for the `queue` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_QUEUE_Command;
```

5.2.11 sema - display information about semaphores

SYNOPSYS:

```
sema [id [id ... ] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Classic API Semaphores currently active in the system.

If invoked with a set of objects ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `sema`:

```
SHLL [/] $ sema
  ID       NAME   ATTR        PRICEIL CURR_CNT HOLDID
------------------------------------------------------------------------------
1a010001   LBIO   PR:BI:IN      0        1     00000000
1a010002   TRmi   PR:BI:IN      0        1     00000000
1a010003   LBI00  PR:BI:IN      0        1     00000000
1a010004   TRia   PR:BI:IN      0        1     00000000
1a010005   TRoa   PR:BI:IN      0        1     00000000
1a010006   TRxa   <assoc.c: BAD NAME>   0    0 09010001
1a010007   LBI01  PR:BI:IN      0        1     00000000
1a010008   LBI02  PR:BI:IN      0        1     00000000
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_SEMA` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_SEMA` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `sema` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_sema(
  int    argc,
  char **argv
);
```

The configuration structure for the `sema` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_SEMA_Command;
```

5.2.12 region - display information about regions

SYNOPSYS:

```
region [id [id ... ] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Classic API Regions currently active in the system.

If invoked with a set of object ids as arguments, then just those object are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of using the `region` command on a system with no user extensions.

```
SHLL [/] $ region
  ID       NAME   ATTR        STARTADDR LENGTH    PAGE_SIZE USED_BLOCKS
----------------------------------------------------------------------------
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_REGION` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_REGION` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `region` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_region(
  int    argc,
  char **argv
);
```

The configuration structure for the `region` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_REGION_Command;
```

5.2.13 part - display information about partitions

SYNOPSYS:

```
part [id [id ... ] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Classic API Partitions currently active in the system.

If invoked with a set of object ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of using the `part` command on a system with no user extensions.

```
SHLL [/] $ part
  ID       NAME   ATTR        STARTADDR LENGTH    BUF_SIZE  USED_BLOCKS
------------------------------------------------------------------------------
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_PART` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_PART` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `part` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_part(
  int    argc,
  char **argv
);
```

The configuration structure for the `part` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_PART_Command;
```

5.2.14 object - display information about rtems objects

SYNOPSYS:

```
object [id [id ...] ]
```

DESCRIPTION:

When invoked with a set of object ids as arguments, then a report on those objects is printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `object`:

```
SHLL [/] $ object 0a010001 1a010002
  ID       NAME   PRIO   STAT   MODES  EVENTS   WAITID  WAITARG  NOTES
------------------------------------------------------------------------------
0a010001   UI1      1   SUSP   P:T:nA  NONE
  ID       NAME   ATTR        PRICEIL CURR_CNT HOLDID
------------------------------------------------------------------------------
1a010002   TRmi   PR:BI:IN      0        1     00000000
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_OBJECT` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_OBJECT` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `object` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_object(
  int    argc,
  char **argv
);
```

The configuration structure for the `object` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_OBJECT_Command;
```

5.2.15 driver - display the rtems device driver table

SYNOPSYS:

```
driver [ major [ major ... ] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of Device Drivers currently active in the system.

If invoked with a set of major numbers as arguments, then just those Device Drivers are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `driver`:

```
SHLL [/] $ driver
  Major      Entry points
-----------------------------------------------------------------------------
  0          init: [0x200256c];  control: [0x20024c8]
             open: [0x2002518];  close: [0x2002504]
             read: [0x20024f0];  write: [0x20024dc]
  1          init: [0x20023fc];  control: [0x2002448]
             open: [0x0];  close: [0x0]
             read: [0x0];  write: [0x0]
SHLL [/] $
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DRIVER` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DRIVER` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `driver` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_driver(
  int    argc,
  char **argv
);
```

The configuration structure for the `driver` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DRIVER_Command;
```

5.2.16 dname - displays information about named drivers

SYNOPSYS:

```
dname
```

DESCRIPTION:

This command XXX

WARNING! XXX This command does not appear to work as of 27 February 2008.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `dname`:

```
EXAMPLE_TBD
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_DNAME` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_DNAME` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `dname` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_dname(
  int    argc,
  char **argv
);
```

The configuration structure for the `dname` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_DNAME_Command;
```

5.2.17 pthread - display information about POSIX threads

SYNOPSYS:

```
pthread [id [id ...] ]
```

DESCRIPTION:

When invoked with no arguments, this command prints information on the set of POSIX API threads currently active in the system.

If invoked with a set of ids as arguments, then just those objects are included in the information printed.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

This command is only available when the POSIX API is configured.

EXAMPLES:

The following is an example of how to use the `task` on an application with four POSIX threads:

```
SHLL [/] $ pthread
  ID       NAME           PRI  STATE MODES   EVENTS    WAITID  WAITARG  NOTES
------------------------------------------------------------------------------
0b010002   Main           133 READY  P:T:nA    NONE   43010001 0x7b1148
0b010003   ISR            133 Wcvar  P:T:nA    NONE   43010003 0x7b1148
0b01000c                  133 READY  P:T:nA    NONE   33010002 0x7b1148
0b01000d                  133 Wmutex P:T:nA    NONE   33010002 0x7b1148
```

CONFIGURATION:

This command is part of the monitor commands which are always available in the shell.

PROGRAMMING INFORMATION:

This command is not directly available for invocation.

6 Network Commands

6.1 Introduction

The RTEMS shell has the following network commands:

- `netstats` - obtain network statistics
- `ifconfig` - configure a network interface
- `route` - show or manipulate the IP routing table

6.2 Commands

This section details the Network Commands available. A subsection is dedicated to each of the commands and describes the behavior and configuration of that command as well as providing an example usage.

6.2.1 netstats - obtain network statistics

SYNOPSYS:

```
netstats [-Aimfpcut]
```

DESCRIPTION:

This command is used to display various types of network statistics. The information displayed can be specified using command line arguments in various combinations. The arguments are interpreted as follows:

-A print All statistics

-i print Inet Routes

-m print MBUF Statistics

-f print IF Statistics

-p print IP Statistics

-c print ICMP Statistics

-u print UDP Statistics

-t print TCP Statistics

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

NONE

EXAMPLES:

The following is an example of how to use `netstats`:

The following is an example of using the `netstats` command to print the IP routing table:

```
[/] $ netstats -i
Destination     Gateway/Mask/Hw    Flags     Refs     Use Expire Interface
default         192.168.1.14       UGS         0        0      0 eth1
192.168.1.0     255.255.255.0      U           0        0      1 eth1
192.168.1.14    00:A0:C8:1C:EE:28  UHL         1        0   1219 eth1
192.168.1.51    00:1D:7E:0C:D0:7C  UHL         0      840   1202 eth1
192.168.1.151   00:1C:23:B2:0F:BB  UHL         1       23   1219 eth1
```

The following is an example of using the `netstats` command to print the MBUF statistics:

```
[/] $ netstats -m
************ MBUF STATISTICS ************
mbufs:2048    clusters: 128    free:  63
drops:   0       waits:   0  drains:   0
      free:1967          data:79         header:2           socket:0
       pcb:0           rtable:0          htable:0           atable:0
    soname:0           soopts:0          ftable:0           rights:0
    ifaddr:0          control:0         oobdata:0
```

The following is an example of using the `netstats` command to print the print the interface statistics:

```
[/] $ netstats -f
************ INTERFACE STATISTICS ************
***** eth1 *****
Ethernet Address: 00:04:9F:00:5B:21
Address:192.168.1.244   Broadcast Address:192.168.1.255   Net mask:255.255.255.0
Flags: Up Broadcast Running Active Multicast
Send queue limit:50   length:1    Dropped:0
      Rx Interrupts:889           Not First:0              Not Last:0
              Giant:0           Non-octet:0
            Bad CRC:0             Overrun:0             Collision:0
      Tx Interrupts:867            Deferred:0        Late Collision:0
   Retransmit Limit:0            Underrun:0            Misaligned:0
```

The following is an example of using the `netstats` command to print the print IP statistics:

```
[/] $ netstats -p
************ IP Statistics ************
             total packets received        894
  packets rcvd for unreachable dest         13
 datagrams delivered to upper level        881
    total ip packets generated here        871
```

The following is an example of using the `netstats` command to print the ICMP statistics:

```
[/] $ netstats -c
************ ICMP Statistics ************
                        Type 0 sent        843
                number of responses        843
                    Type 8 received        843
```

The following is an example of using the `netstats` command to print the UDP statistics:

```
[/] $ netstats -u
************ UDP Statistics ************
```

The following is an example of using the `netstats` command to print the TCP statistics:

```
[/] $ netstats -t
************ TCP Statistics ************
               connections accepted          1
            connections established          1
     segs where we tried to get rtt         34
                 times we succeeded         35
                  delayed acks sent          2
                 total packets sent         37
                  data packets sent         35
                    data bytes sent       2618
              ack-only packets sent          2
             total packets received         47
       packets received in sequence         12
         bytes received in sequence        307
                   rcvd ack packets         35
           bytes acked by rcvd acks       2590
      times hdr predict ok for acks         27
 times hdr predict ok for data pkts         10
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_NETSTATS` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_NETSTATS` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `netstats` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_netstats(
  int    argc,
  char **argv
);
```

The configuration structure for the `netstats` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_NETSTATS_Command;
```

6.2.2 ifconfig - configure a network interface

SYNOPSYS:

```
ifconfig
ifconfig interface
ifconfig interface [up|down]
ifconfig interface [netmask|pointtopoint|broadcast] IP
```

DESCRIPTION:

This command may be used to display information about the network interfaces in the system or configure them.

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

Just like its counterpart on GNU/Linux and BSD systems, this command is complicated. More example usages would be a welcome submission.

EXAMPLES:

The following is an example of how to use `ifconfig`:

```
 ************ INTERFACE STATISTICS ************
***** eth1 *****
Ethernet Address: 00:04:9F:00:5B:21
Address:192.168.1.244   Broadcast Address:192.168.1.255   Net mask:255.255.255.0
Flags: Up Broadcast Running Active Multicast
Send queue limit:50   length:1    Dropped:0
      Rx Interrupts:5391          Not First:0              Not Last:0
              Giant:0             Non-octet:0
            Bad CRC:0               Overrun:0             Collision:0
      Tx Interrupts:5256           Deferred:0        Late Collision:0
   Retransmit Limit:0              Underrun:0            Misaligned:0
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_IFCONFIG` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_IFCONFIG` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `ifconfig` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_ifconfig(
  int    argc,
  char **argv
);
```

The configuration structure for the `ifconfig` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_IFCONFIG_Command;
```

6.2.3 route - show or manipulate the ip routing table

SYNOPSYS:

```
route [subcommand] [args]
```

DESCRIPTION:

This command is used to display and manipulate the routing table. When invoked with no arguments, the current routing information is displayed. When invoked with the subcommands `add` or `del`, then additional arguments must be provided to describe the route.

Command templates include the following:

```
route [add|del] -net IP_ADDRESS gw GATEWAY_ADDRESS [netmask MASK]
route [add|del] -host IP_ADDRESS gw GATEWAY_ADDRES [netmask MASK]
```

When not provided the netmask defaults to `255.255.255.0`

EXIT STATUS:

This command returns 0 on success and non-zero if an error is encountered.

NOTES:

Just like its counterpart on GNU/Linux and BSD systems, this command is complicated. More example usages would be a welcome submission.

EXAMPLES:

The following is an example of how to use `route` to display, add, and delete a new route:

```
[/] $ route
Destination     Gateway/Mask/Hw    Flags     Refs     Use Expire Interface
default         192.168.1.14       UGS         0        0      0 eth1
192.168.1.0     255.255.255.0      U           0        0      1 eth1
192.168.1.14    00:A0:C8:1C:EE:28  UHL         1        0   1444 eth1
192.168.1.51    00:1D:7E:0C:D0:7C  UHL         0    10844   1202 eth1
192.168.1.151   00:1C:23:B2:0F:BB  UHL         2       37   1399 eth1
[/] $  route add -net 192.168.3.0 gw 192.168.1.14
[/] $ route
Destination     Gateway/Mask/Hw    Flags     Refs     Use Expire Interface
default         192.168.1.14       UGS         0        0      0 eth1
192.168.1.0     255.255.255.0      U           0        0      1 eth1
192.168.1.14    00:A0:C8:1C:EE:28  UHL         2        0   1498 eth1
192.168.1.51    00:1D:7E:0C:D0:7C  UHL         0    14937   1202 eth1
192.168.1.151   00:1C:23:B2:0F:BB  UHL         2       96   1399 eth1
192.168.3.0     192.168.1.14       UGS         0        0      0 eth1
[/] $ route del -net 192.168.3.0 gw 192.168.1.14
[/] $ route
Destination     Gateway/Mask/Hw    Flags     Refs     Use Expire Interface
default         192.168.1.14       UGS         0        0      0 eth1
192.168.1.0     255.255.255.0      U           0        0      1 eth1
192.168.1.14    00:A0:C8:1C:EE:28  UHL         1        0   1498 eth1
192.168.1.51    00:1D:7E:0C:D0:7C  UHL         0    15945   1202 eth1
192.168.1.151   00:1C:23:B2:0F:BB  UHL         2      117   1399 eth1
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define `CONFIGURE_SHELL_COMMAND_ROUTE` to have this command included.

This command can be excluded from the shell command set by defining `CONFIGURE_SHELL_NO_COMMAND_ROUTE` when all shell commands have been configured.

PROGRAMMING INFORMATION:

The `route` is implemented by a C language function which has the following prototype:

```
int rtems_shell_rtems_main_route(
  int    argc,
  char **argv
);
```

The configuration structure for the `route` has the following prototype:

```
extern rtems_shell_cmd_t rtems_shell_ROUTE_Command;
```

Function and Variable Index

C

R

Concept Index

Command Index

T

U

W