



RTEMS Shell Documentation

Release 4.11.0

Oct 31, 2016

CONTENTS

I	RTEMS Shell User's Guide	1
1	Preface	3
1.1	Acknowledgements	4
2	Configuration and Initialization	5
2.1	Introduction	5
2.2	Configuration	5
2.3	Initialization	7
2.4	Access Control	8
2.5	Functions	9
3	General Commands	11
3.1	Introduction	11
3.2	Commands	11
4	File and Directory Commands	27
4.1	Introduction	27
4.2	Commands	28
5	Memory Commands	63
5.1	Introduction	63
5.2	Commands	63
6	RTEMS Specific Commands	73
6.1	Introduction	73
6.2	Commands	73
7	Network Commands	93
7.1	Introduction	93
7.2	Commands	93
8	Function and Variable Index	103
9	Concept Index	105
	Index	107

Part I

RTEMS SHELL USER'S GUIDE

COPYRIGHT (c) 1988 - 2015.

On-Line Applications Research Corporation (OAR).

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

The RTEMS Project is hosted at <http://www.rtems.org/>. Any inquiries concerning RTEMS, its related support components, or its documentation should be directed to the Community Project hosted at <http://www.rtems.org/>.

RTEMS Online Resources

Home	https://www.rtems.org/
Developers	https://devel.rtems.org/
Documentation	https://docs.rtems.org/
Bug Reporting	https://devel.rtems.org/query
Mailing Lists	https://lists.rtems.org/
Git Repositories	https://git.rtems.org/

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:

```
1 Welcome to rtems-4.10.99.0(SPARC/w/FPU/sis)
2 COPYRIGHT (c) 1989-2011.
3 On-Line Applications Research Corporation (OAR).
4 Login into RTEMS
5 login: rtems
6 Password:
7 RTEMS SHELL (Ver.1.0-FRC):/dev/console. Feb 28 2008. 'help' to list commands.
8 SHLL [/] $ cat /etc/passwd
9 root:*:0:0:root:::/bin/sh
10 rtems:*:1:1:RTEMS Application:::/bin/sh
11 tty!:2:2:tty owner:::/bin/false
12 SHLL [/] $ ls /dev
13 -rwxr-xr-x 1 rtems root          0 Jan 01 00:00 console
14 -rwxr-xr-x 1 root  root          0 Jan 01 00:00 console_b
15 2 files 0 bytes occupied
16 SHLL [/] $ stackuse
17 Stack usage by thread
18 ID      NAME    LOW          HIGH        CURRENT    AVAILABLE    USED
19 0x09010001 IDLE 0x023d89a0 - 0x023d99af 0x023d9760    4096      608
20 0x0a010001 UI1  0x023d9f30 - 0x023daf3f 0x023dad18    4096     1804
21 0x0a010002 SHLL 0x023db4c0 - 0x023df4cf 0x023de9d0   16384     6204
22 0xffffffff INTR 0x023d2760 - 0x023d375f 0x00000000    4080      316
23 SHLL [/] $ mount -L
24 File systems: msdos
25 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 accessed 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 filesystems such as NFS and TFTP, the standard free RTEMS provides a robust 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.

1.1 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 (c) 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>. This notice shall appear on any product containing this material.

CONFIGURATION AND INITIALIZATION

2.1 Introduction

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

2.2 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, formatting, etc.) or network related, then the following is all that is required:

```
1 #define CONFIGURE_SHELL_COMMANDS_INIT
2 #define CONFIGURE_SHELL_COMMANDS_ALL
3 #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:

```
1 #define CONFIGURE_SHELL_COMMANDS_INIT
2 #define CONFIGURE_SHELL_COMMANDS_ALL
3 #define CONFIGURE_SHELL_MOUNT_MSDOS
4 #define CONFIGURE_SHELL_MOUNT_NFS
5 #include <rtems/shellconfig.h>
```

2.2.1 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.

2.2.2 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`.

```

1 #include <rtems/shell.h>
2 int main_usercmd(int argc, char **argv)
3 {
4     int i;
5     printf( "UserCommand: argc=%d\n", argc );
6     for (i=0 ; i<argc ; i++ )
7         printf( "argv[%d]= %s\n", i, argv[i] );
8     return 0;
9 }
10 rtems_shell_cmd_t Shell_USERCMD_Command = {
11     "usercmd",                /* name */
12     "usercmd n1 \[n2 \[n3...]", /* usage */
13     "user",                   /* topic */
14     main_usercmd,              /* command */
15     NULL,                     /* alias */
16     NULL                       /* next */
17 };
18 rtems_shell_alias_t Shell_USERECHO_Alias = {
19     "usercmd",                /* command */
20     "userecho"                /* alias */
21 };
22 #define CONFIGURE_SHELL_USER_COMMANDS &Shell_USERCMD_Command
23 #define CONFIGURE_SHELL_USER_ALIASES &Shell_USERECHO_Alias
24 #define CONFIGURE_SHELL_COMMANDS_INIT
25 #define CONFIGURE_SHELL_COMMANDS_ALL
26 #define CONFIGURE_SHELL_MOUNT_MSDOS
27 #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.

2.3 Initialization

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

2.3.1 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.

```
1 #include <rtems/shell.h>
2 void start_shell(void)
3 {
4     printf(" =====\n");
5     printf(" starting shell\n");
6     printf(" =====\n");
7     rtems_shell_init(
8         "SHLL", /* task name */
9         RTEMS_MINIMUM_STACK_SIZE * 4, /* task stack size */
10        100, /* task priority */
11        "/dev/console", /* device name */
12        false, /* run forever */
13        true, /* wait for shell to terminate */
14        rtems_shell_login_check /* login check function,
15        use NULL to disable a login check */
16    );
17 }
```

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.

2.3.2 Attached to a Socket

TBD

2.4 Access Control

2.4.1 Login Checks

Login checks are optional for the RTEMS shell and can be configured via a login check handler passed to `rtems_shell_init()`. One login check handler is `rtems_shell_login_check()`.

2.4.2 Configuration Files

The following files are used by the login check handler `rtems_shell_login_check()` to validate a passphrase for a user and to set up the user environment for the shell command execution.

/etc/passwd The format for each line is

```
1 user_name:password:UID:GID:GECOS:directory:shell
```

with colon separated fields. For more information refer to the Linux `PASSWD(5)` man page. Use a password of `*` to disable the login of the user. An empty password allows login without a password for this user. In contrast to standard UNIX systems, this file is only readable and writeable for the user with an UID of zero by default. The directory is used to perform a filesystem change root operation in `rtems_shell_login_check()` in contrast to a normal usage as the HOME directory of the user. The *default* content is:

```
1 root::0:0:::
```

so there is *no password required* for the root user.

/etc/group The format for each line is:

```
1 group_name:password:GID:user_list
```

with colon separated fields. The `user_list` is comma separated. For more information refer to the Linux `GROUP(5)` man page. In contrast to standard UNIX systems, this file is only readable and writeable for the user with an UID of zero by default. The default content is

```
1 root::0:
```

2.4.3 Command Visibility and Execution Permission

Each command has:

- an owner,
- a group, and
- a read permission flag for the owner, the group and all other users, and
- an execution permission flag for the owner, the group and all other users.

The read and write permission flags are stored in the command mode. The read permission flags determine the visibility of the command for the current user. The execution permission flags determine the ability to execute a command for the current user. These command properties can be displayed and changed with the:

- `cmdls`,
- `cmdchown`, and
- `cmdchmod`

commands. The access is determined by the effective UID, the effective GID and the supplementary group IDs of the current user and follows the standard filesystem access procedure.

2.4.4 Add CRYPT(3) Formats

By default the `crypt_r()` function used by `rtems_shell_login_check()` supports only plain text passphrases. Use `crypt_add_format()` to add more formats. The following formats are available out of the box:

- `crypt_md5_format`,
- `crypt_sha256_format`, and
- `crypt_sha512_format`.

An example follows:

```

1 #include <crypt.h>
2 void add_formats( void )
3 {
4     crypt_add_format( &crypt_md5_format );
5     crypt_add_format( &crypt_sha512_format );
6 }
```

2.5 Functions

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

2.5.1 `rtems_shell_init` - Initialize the shell

CALLING SEQUENCE:

```

1 rtems_status_code rtems_shell_init(
2     const char      *task_name,
3     size_t          task_stacksize,
4     rtems_task_priority task_priority,
5     const char      *devname,
6     bool             forever,
7     bool             wait,
8     rtems_login_check login_check
9 );
```

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.

There is one POSIX key necessary for all shell instances together and one POSIX key value pair per instance. You should make sure that your RTEMS configuration accounts for these resources.

2.5.2 `rtems_shell_login_check` - Default login check handler

CALLING SEQUENCE:

```
1 bool rtems_shell_login_check(  
2     const char *user,  
3     const char *passphrase  
4 );
```

DIRECTIVE STATUS CODES:

true - login is allowed, and false - otherwise.

DESCRIPTION:

This function checks if the specified passphrase is valid for the specified user.

NOTES:

As a side-effect if the specified passphrase is valid for the specified user, this function:

- performs a filesystem change root operation to the directory of the specified user if the directory path is non-empty,
- changes the owner of the current shell device to the UID of the specified user,
- sets the real and effective UID of the current user environment to the UID of the specified user,
- sets the real and effective GID of the current user environment to the GID of the specified user, and
- sets the supplementary group IDs of the current user environment to the supplementary group IDs of the specified user.

In case the filesystem change root operation fails, then the environment setup is aborted and false is returned.

GENERAL COMMANDS

3.1 Introduction

The RTEMS shell has the following general commands:

- *help* (page 12) - Print command help
- *alias* (page 13) - Add alias for an existing command
- *cmds* (page 14) - List commands
- *cmdchown* (page 14) - Change user or owner of commands
- *cmdchmod* (page 15) - Change mode of commands
- *date* (page 16) - Print or set current date and time
- *echo* (page 17) - Produce message in a shell script
- *sleep* (page 18) - Delay for a specified amount of time
- *id* (page 19) - show uid gid euid and egid
- *tty* (page 20) - show ttyname
- *whoami* (page 21) - print effective user id
- *getenv* (page 21) - print environment variable
- *setenv* (page 22) - set environment variable
- *unsetenv* (page 23) - unset environment variable
- *time* (page 24) - time command execution
- *logoff* (page 25) - logoff from the system
- *rtc* (page 25) - RTC driver configuration
- *exit* (page 26) - alias for logoff command

3.2 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.

3.2.1 help - Print command help

SYNOPSIS:

```
1 help misc
```

DESCRIPTION:

This command prints the command help. Help without arguments prints a list of topics and help with a topic prints the help for that topic.

EXIT STATUS:

This command returns 0.

NOTES:

The help print will break the output up based on the environment variable `SHELL_LINES`. If this environment variable is not set the default is 16 lines. If set the number of lines is set to that the value. If the shell lines is set 0 there will be no break.

EXAMPLES:

The following is an example of how to use alias:

```
1 SHLL [/] $ help
2 help: ('r' repeat last cmd - 'e' edit last cmd)
3 TOPIC? The topics are
4 mem, misc, files, help, rtems, network, monitor
5 SHLL [/] $ help misc
6 help: list for the topic 'misc'
7 alias      - alias old new
8 time       - time command [arguments...]
9 joel       - joel [args] SCRIPT
10 date       - date [YYYY-MM-DD HH:MM:SS]
11 echo       - echo [args]
12 sleep      - sleep seconds [nanoseconds]
13 id         - show uid, gid, euid, and egid
14 tty        - show ttyname
15 whoami     - show current user
16 logoff     - logoff from the system
17 setenv     - setenv [var] [string]
18 getenv     - getenv [var]
19 unsetenv   - unsetenv [var]
20 umask      - umask [new_umask]
21 Press any key to continue...
22 rtc        - real time clock read and set
23 SHLL [/] $ setenv SHELL_ENV 0
24 SHLL [/] $ help misc
25 help: list for the topic 'misc'
26 alias      - alias old new
27 time       - time command [arguments...]
28 joel       - joel [args] SCRIPT
29 date       - date [YYYY-MM-DD HH:MM:SS]
30 echo       - echo [args]
31 sleep      - sleep seconds [nanoseconds]
32 id         - show uid, gid, euid, and egid
33 tty        - show ttyname
34 whoami     - show current user
```



```

35 logoff      - logoff from the system
36 setenv      - setenv [var] [string]
37 getenv      - getenv [var]
38 unsetenv    - unsetenv [var]
39 umask       - umask [new_umask]
40 rtc         - real time clock read and set

```

CONFIGURATION:

This command has no configuration.

3.2.2 alias - add alias for an existing command**SYNOPSIS:**

```
1 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:

```

1 SHLL [/] $ me
2 shell:me command not found
3 SHLL [/] $ alias whoami me
4 SHLL [/] $ me
5 rtems
6 SHLL [/] $ whoami
7 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:

```

1 int rtems_shell_rtems_main_alias(
2     int  argc,
3     char **argv
4 );

```

The configuration structure for the alias has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_ALIAS_Command;
```

3.2.3 cmdls - List commands

SYNOPSIS:

```
1 cmdls COMMAND...
```

DESCRIPTION:

This command lists the visible commands of the command set.

EXIT STATUS:

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

NOTES:

The current user must have read permission to list a command.

EXAMPLES:

The following is an example of how to use cmdls:

```
1 SHLL [/] # cmdls help shutdown
2 r-xr-xr-x    0    0 help
3 r-x-----    0    0 shutdown
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

The configuration structure for the cmdls has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_CMDLS_Command;
```

3.2.4 cmdchown - Change user or owner of commands

SYNOPSIS:

```
1 cmdchown [OWNER][:[GROUP]] COMMAND...
```

DESCRIPTION:

This command changes the user or owner of a command.

EXIT STATUS:

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

NOTES:

The current user must have an UID of zero or be the command owner to change the owner or group.

EXAMPLES:

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

```
1 [/] # cmdls help
2 r-xr-xr-x    0    0 help
3 [/] # cmdchown 1:1 help
4 [/] # cmdls help
5 r--r--r--    1    1 help
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```
1 extern rtems_shell_cmd_t rtems_shell_CMDCHOWN_Command;
```

3.2.5 cmdchmod - Change mode of commands

SYNOPSIS:

```
1 cmdchmod OCTAL-MODE COMMAND...
```

DESCRIPTION:

This command changes the mode of a command.

EXIT STATUS:

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

NOTES:

The current user must have an UID of zero or be the command owner to change the mode.

EXAMPLES:

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

```
1 [/] # cmdls help
2 r-xr-xr-x    0    0 help
3 [/] # cmdchmod 544 help
4 [/] # cmdls help
5 r-xr--r--    0    0 help
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```
1 extern rtems_shell_cmd_t rtems_shell_CMDCHMOD_Command;
```

3.2.6 date - print or set current date and time

SYNOPSIS:

```
1 date
2 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:

None.

EXAMPLES:

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

```
1 SHLL [/] $ date
2 Fri Jan 1 00:00:09 1988
3 SHLL [/] $ date 2008-02-29 06:45:32
4 SHLL [/] $ date
5 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:

```
1 int rtems_shell_rtems_main_date(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the date has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_DATE_Command;
```

3.2.7 echo - produce message in a shell script

SYNOPSIS:

```
1 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.

Odigits 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 (*Odigits*) differs from the C language 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:

```
1 SHLL [/] $ echo a b c
2 a b c
3 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:

```
1 int rtems_shell_rtems_main_echo(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the echo has the following prototype:

```
1 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.

3.2.8 sleep - delay for a specified amount of time**SYNOPSIS:**

```
1 sleep seconds
2 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`:

```
1 SHLL [/] $ sleep 10
2 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:

```
1 int rtems_shell_rtems_main_sleep(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the sleep has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_SLEEP_Command;
```

3.2.9 id - show uid gid euid and egid**SYNOPSIS:**

```
1 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`:

```
1 SHLL [/] # id
2 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:

```
1 SHLL [/] # id
2 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:

```
1 int rtems_shell_rtems_main_id(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_ID_Command;
```

3.2.10 `tty` - show ttyname

SYNOPSIS:

```
1 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`:

```
1 SHLL [/] $ tty  
2 /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:

```
1 int rtems_shell_rtems_main_tty(  
2     int    argc,  
3     char **argv  
4 );
```

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


```
1 extern rtems_shell_cmd_t rtems_shell_TTY_Command;
```

3.2.11 whoami - print effective user id

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ whoami
2 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:

```
1 int rtems_shell_rtems_main_whoami(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the whoami has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_WHOAMI_Command;
```

3.2.12 getenv - print environment variable

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ getenv BASEPATH
2 /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:

```
1 int rtems_shell_rtems_main_getenv(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_GETENV_Command;
```

3.2.13 setenv - set environment variable

SYNOPSIS:

```
1 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:

```
1 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:

```
1 int rtems_shell_rtems_main_setenv(  
2     int    argc,  
3     char **argv  
4 );
```

The configuration structure for the setenv has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_SETENV_Command;
```

3.2.14 unsetenv - unset environment variable**SYNOPSIS:**

```
1 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:

```
1 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:

```
1 int rtems_shell_rtems_main_unsetenv(  
2     int    argc,  
3     char **argv  
4 );
```

The configuration structure for the unsetenv has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_UNSETENV_Command;
```

3.2.15 time - time command execution

SYNOPSIS:

```
1 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:

```
1 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:

```
1 int rtems_shell_rtems_main_time(  
2     int    argc,  
3     char **argv  
4 );
```

The configuration structure for the time has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_TIME_Command;
```

3.2.16 logoff - logoff from the system

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ logoff
2 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:

```
1 int rtems_shell_rtems_main_logoff(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the logoff has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_LOGOFF_Command;
```

3.2.17 rtc - RTC driver configuration

SYNOPSIS:

```
1 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.

3.2.18 `exit` - exit the shell

SYNOPSIS:

```
1 exit
```

DESCRIPTION:

This command causes the shell interpreter to exit.

EXIT STATUS:

This command does not return.

NOTES:

In contrast to *logoff* - *logoff from the system*, this command is built into the shell interpreter loop.

EXAMPLES:

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

```
1 SHLL [/] $ exit
2 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.

FILE AND DIRECTORY COMMANDS

4.1 Introduction

The RTEMS shell has the following file and directory commands:

- *blksync* (page 28) - sync the block driver
- *cat* (page 29) - display file contents
- *cd* (page 29) - alias for *chdir*
- *chdir* (page 30) - change the current directory
- *chmod* (page 31) - change permissions of a file
- *chroot* (page 33) - change the root directory
- *cp* (page 34) - copy files
- *dd* (page 36) - convert and copy a file
- *debugrfs* (page 39) - debug RFS file system
- *df* (page 41) - display file system disk space usage
- *dir* (page 42) - alias for *ls* (page 47)
- *fdisk* (page 42) - format disks
- *hexdump* (page 43) - format disks
- *ln* (page 46) - make links
- *ls* (page 47) - list files in the directory
- *md5* (page 48) - display file system disk space usage
- *mkdir* (page 49) - create a directory
- *mkdos* (page 50) - DOSFS disk format
- *mknod* (page 51) - make device special file
- *mkrfs* (page 53) - format RFS file system
- *mount* (page 54) - mount disk
- *mv* (page 56) - move files
- *pwd* (page 57) - print work directory

- *rmdir* (page 58) - remove empty directories
- *rm* (page 59) - remove files
- *umask* (page 60) - Set file mode creation mask
- *umount* (page 60) - unmount disk

4.2 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.

4.2.1 blksync - sync the block driver

SYNOPSIS:

```
1 blksync driver
```

DESCRIPTION:

This command issues a block driver sync call to the driver. The driver is a path to a device node. The sync call will flush all pending writes in the cache to the media and block until the writes have completed.

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:

```
1 blksync /dev/hda1
```

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:

```
1 int rtems_shell_rtems_main_blksync(
2     int    argc,
3     char **argv
4 );
```


The configuration structure for the blksync has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_BLKSYNC_Command;
```

4.2.2 cat - display file contents

SYNOPSIS:

```
1 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:

```
1 SHLL [/] # cat /etc/passwd
2 root:*:0:0:root:::/bin/sh
3 rtems:*:1:1:RTEMS Application:::/bin/sh
4 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:

```
1 int rtems_shell_rtems_main_cat(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the cat has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_CAT_Command;
```

4.2.3 cd - alias for chdir

SYNOPSIS:

```
1 cd directory
```

DESCRIPTION:

This command is an alias or alternate name for the `chdir`. See *ls - list files in the directory* 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`:

```
1 SHLL [/] $ cd etc
2 SHLL [/etc] $ cd /
3 SHLL [/] $ cd /etc
4 SHLL [/etc] $ pwd
5 /etc
6 SHLL [/etc] $ cd /
7 SHLL [/] $ pwd
8 /
9 SHLL [/] $ cd etc
10 SHLL [/etc] $ cd ..
11 SHLL [/] $ pwd
12 /
```

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:

```
1 int rtems_shell_rtems_main_cd(
2     int  argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_CD_Command;
```

4.2.4 `chdir` - change the current directory

SYNOPSIS:

```
1 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`:

```
1 SHLL [/] $ pwd
2 /
3 SHLL [/] $ chdir etc
4 SHLL [/etc] $ pwd
5 /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:

```
1 int rtems_shell_rtems_main_chdir(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_CHDIR_Command;
```

4.2.5 chmod - change permissions of a file

SYNOPSIS:

```
1 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`:

```

1 SHLL [/] # cd etc
2 SHLL [/etc] # ls
3 -rw-r--r-- 1 root root 102 Jan 01 00:00 passwd
4 -rw-r--r-- 1 root root 42 Jan 01 00:00 group
5 -rw-r--r-- 1 root root 30 Jan 01 00:00 issue
6 -rw-r--r-- 1 root root 28 Jan 01 00:00 issue.net
7 4 files 202 bytes occupied
8 SHLL [/etc] # chmod 0777 passwd
9 SHLL [/etc] # ls
10 -rwxrwxrwx 1 root root 102 Jan 01 00:00 passwd
11 -rw-r--r-- 1 root root 42 Jan 01 00:00 group
12 -rw-r--r-- 1 root root 30 Jan 01 00:00 issue
13 -rw-r--r-- 1 root root 28 Jan 01 00:00 issue.net
14 4 files 202 bytes occupied
15 SHLL [/etc] # chmod 0322 passwd
16 SHLL [/etc] # ls
17 --wx-w--w- 1 nouser root 102 Jan 01 00:00 passwd
18 -rw-r--r-- 1 nouser root 42 Jan 01 00:00 group
19 -rw-r--r-- 1 nouser root 30 Jan 01 00:00 issue
20 -rw-r--r-- 1 nouser root 28 Jan 01 00:00 issue.net
21 4 files 202 bytes occupied
22 SHLL [/etc] # chmod 0644 passwd
23 SHLL [/etc] # ls
24 -rw-r--r-- 1 root root 102 Jan 01 00:00 passwd
25 -rw-r--r-- 1 root root 42 Jan 01 00:00 group
26 -rw-r--r-- 1 root root 30 Jan 01 00:00 issue
27 -rw-r--r-- 1 root root 28 Jan 01 00:00 issue.net
28 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:

```

1 int rtems_shell_rtems_main_chmod(
2     int argc,
3     char **argv
4 );

```

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

```
1 extern rtems_shell_cmd_t rtems_shell_CHMOD_Command;
```

4.2.6 chroot - change the root directory

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ cat passwd
2 cat: passwd: No such file or directory
3 SHLL [/] $ chroot etc
4 SHLL [/] $ cat passwd
5 root:*:0:0:root:::/bin/sh
6 rtems:*:1:1:RTEMS Application:::/bin/sh
7 tty!:2:2:tty owner:::/bin/false
8 SHLL [/] $ cat /etc/passwd
9 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. Additional to that you have to add one POSIX key value pair for each thread where you want to use the command.

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:

```
1 int rtems_shell_rtems_main_chroot(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_CHROOT_Command;
```

4.2.7 cp - copy files

SYNOPSIS:

```
1 cp [-R [-H | -L | -P]] [-f | -i] [-pv] src target
2 cp [-R [-H | -L] ] [-f | -i] [-NpV] 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:

```

1 SHLL [/] # cat joel
2 cat: joel: No such file or directory
3 SHLL [/] # cp etc/passwd joel
4 SHLL [/] # cat joel
5 root:*:0:0:root:::/bin/sh
6 rtems:*:1:1:RTEMS Application:::/bin/sh
7 tty!:2:2:tty owner:::/bin/false
8 SHLL [/] # ls
9 drwxr-xr-x  1  root  root      536 Jan 01 00:00 dev/
10 drwxr-xr-x  1  root  root     1072 Jan 01 00:00 etc/
11 -rw-r--r--  1  root  root      102 Jan 01 00:00 joel
12 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:

```

1 SHLL [/] # mkdir tmp
2 SHLL [/] # ls tmp
3 0 files 0 bytes occupied
4 SHLL [/] # cp /etc/passwd tmp
5 SHLL [/] # ls /tmp
6 -rw-r--r--  1  root  root      102 Jan 01 00:01 passwd
7 1 files 102 bytes occupied
8 SHLL [/] # cp /etc/passwd /etc/group /tmp
9 SHLL [/] # ls /tmp
10 -rw-r--r--  1  root  root      102 Jan 01 00:01 passwd
11 -rw-r--r--  1  root  root       42 Jan 01 00:01 group

```

```

12 2 files 144 bytes occupied
13 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:

```

1 int rtems_shell_main_cp(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the cp has the following prototype:

```

1 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.

4.2.8 dd - convert and copy a file**SYNOPSIS:**

```

1 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 recommended 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`:

```
1 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:

```
1 int rtems_shell_rtems_main_dd(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_DD_Command;
```

4.2.9 debugrfs - debug RFS file system

SYNOPSIS:

```
1 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:

```
1 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:

```
1 int rtems_shell_rtems_main_debugrfs(  
2     int    argc,  
3     char **argv  
4 );
```

The configuration structure for debugrfs has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_DEBUGRFS_Command;
```

4.2.10 df - display file system disk space usage

SYNOPSIS:

```
1 df [-h] [-B block_size]
```

DESCRIPTION:

This command print disk space usage for mounted file systems.

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 df:

```
1 SHLL [/] $ df -B 4K
2 Filesystem      4K-blocks      Used    Available      Use%    Mounted on
3 /dev/rda                124          1         124          0%    /mnt/ramdisk
4 SHLL [/] $ df
5 Filesystem      1K-blocks      Used    Available      Use%    Mounted on
6 /dev/rda                495          1         494          0%    /mnt/ramdisk
7 SHLL [/] $ df -h
8 Filesystem      Size           Used    Available      Use%    Mounted on
9 /dev/rda        495K            1K         494K          0%    /mnt/ramdisk
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```
1 int rtems_shell_main_df(
2     int  argc,
3     char **argv
4 );
```

The configuration structure for the df has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_DF_Command;
```

4.2.11 dir - alias for ls

SYNOPSIS:

```
1 dir [dir]
```

DESCRIPTION:

This command is an alias or alternate name for the `ls`. See *ls - list files in the directory* 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`:

```
1 SHLL [/] $ dir
2 drwxr-xr-x  1  root  root          536 Jan 01 00:00 dev/
3 drwxr-xr-x  1  root  root        1072 Jan 01 00:00 etc/
4 2 files 1608 bytes occupied
5 SHLL [/] $ dir etc
6 -rw-r--r--  1  root  root         102 Jan 01 00:00 passwd
7 -rw-r--r--  1  root  root          42 Jan 01 00:00 group
8 -rw-r--r--  1  root  root          30 Jan 01 00:00 issue
9 -rw-r--r--  1  root  root          28 Jan 01 00:00 issue.net
10 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:

```
1 int rtems_shell_rtems_main_dir(
2     int  argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_DIR_Command;
```

4.2.12 fdisk - format disk

SYNOPSIS:

```
1 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.

4.2.13 hexdump - ascii/dec/hex/octal dump

SYNOPSIS:

```
1 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, zero-filled, 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" conversion 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:

```
1 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:

```
1 int rtems_shell_rtems_main_hexdump(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the hexdump has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_HEXDUMP_Command;
```

4.2.14 ln - make links

SYNOPSIS:

```
1 ln [-fhinsv] source_file [target_file]
2 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:

```
1 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:

```
1 int rtems_shell_rtems_main_ln(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 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.

4.2.15 ls - list files in the directory

SYNOPSIS:

```
1 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`:

```

1 SHLL [/] $ ls
2 drwxr-xr-x  1  root  root          536 Jan 01 00:00 dev/
3 drwxr-xr-x  1  root  root        1072 Jan 01 00:00 etc/
4 2 files 1608 bytes occupied
5 SHLL [/] $ ls etc
6 -rw-r--r--  1  root  root         102 Jan 01 00:00 passwd
7 -rw-r--r--  1  root  root          42 Jan 01 00:00 group
8 -rw-r--r--  1  root  root          30 Jan 01 00:00 issue
9 -rw-r--r--  1  root  root          28 Jan 01 00:00 issue.net
10 4 files 202 bytes occupied
11 SHLL [/] $ ls dev etc
12 -rwxr-xr-x  1  rtems  root          0 Jan 01 00:00 console
13 -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:

```

1 int rtems_shell_rtems_main_ls(
2     int  argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_LS_Command;

```

4.2.16 md5 - compute the Md5 hash of a file or list of files

SYNOPSIS:

```

1 md5 <files>

```

DESCRIPTION:

This command prints the MD5 of a file. You can provide one or more files on the command line and a hash for each file is printed in a single line of output.

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 md5:

```
1 SHLL [/] $ md5 shell-init
2 MD5 (shell-init) = 43b4d2e71b47db79eae679a2efeacf31
```

CONFIGURATION:

This command is included in the default shell command set. When building a custom command set, define “CONFIGURE_SHELL_COMMAND_MD5” to have this command included.

This command can be excluded from the shell command set by defining CONFIGURE_SHELL_NO_COMMAND_MD5 when all shell commands have been configured.

PROGRAMMING INFORMATION:

The md5 is implemented by a C language function which has the following prototype:

```
1 int rtems_shell_main_md5(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the md5 has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_MD5_Command;
```

4.2.17 mkdir - create a directory

SYNOPSIS:

```
1 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`:

```

1 SHLL [/] # ls
2 drwxr-xr-x  1  root  root          536 Jan 01 00:00 dev/
3 drwxr-xr-x  1  root  root       1072 Jan 01 00:00 etc/
4 2 files 1608 bytes occupied
5 SHLL [/] # mkdir joel
6 SHLL [/] # ls joel
7 0 files 0 bytes occupied
8 SHLL [/] # cp etc/passwd joel
9 SHLL [/] # ls joel
10 -rw-r--r--  1  root  root          102 Jan 01 00:02 passwd
11 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:

```

1 int rtems_shell_rtems_main_mkdir(
2     int  argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_MKDIR_Command;

```

4.2.18 mkdos - DOSFS file system format

SYNOPSIS:

```

1 mkdos [-V label] [-s sectors/cluster] [-r size] [-v] path

```

DESCRIPTION:

This command formats a block device entry with the DOSFS file system.

-V *label* Specify the volume label.

-s *sectors/cluster* Specify the number of sectors per cluster.

-r *size* Specify the number of entries in the root directory.

-v Enable verbose output mode.

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 mkdos:

```
1 SHLL [/] $ mkdos /dev/rda1
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```
1 int rtems_shell_rtems_main_mkdos(
2     int     argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_MKDOS_Command;
```

4.2.19 mknod - make device special file

SYNOPSIS:

```
1 mknod [-rR] [-F fmt] [-g gid] [-m mode] [-u uid] name [c | b] [driver | major] minor
2 mknod [-rR] [-F fmt] [-g gid] [-m mode] [-u uid] name [c | b] major unit subunit
3 mknod [-rR] [-g gid] [-m mode] [-u uid] name [c | b] number
4 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:

```
1 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:

```
1 int rtems_shell_rtems_main_mknod(
2     int    argc,
3     char **argv
4 );
```


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

```
1 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.

4.2.20 `mkrf`s - format RFS file system

SYNOPSIS:

```
1 mkrf[s] [-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 `mkrf`s:

```
1 SHLL [/] $ mkrf[s] /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 `mkrf`s command is implemented by a C language function which has the following prototype:

```
1 int rtems_shell_rtems_main_mkrf(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for `mkrf`s has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_MKRFS_Command;
```

4.2.21 mount - mount disk

SYNOPSIS:

```
1 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:

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

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

```
1 $ mount -t nfs foo:/bar /nfs
```

Mount the TFTP file system on '/tftp':

```
1 $ mount -t tftp /tftp
```

To access the TFTP files on server '10.10.10.10': .. code-block:: shell

```
$ 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:

```
1 #define CONFIGURE_SHELL_MOUNT_MSDOS
2 #ifdef RTEMS_NETWORKING
3 #define CONFIGURE_SHELL_MOUNT_TFTP
4 #define CONFIGURE_SHELL_MOUNT_FTP
5 #define CONFIGURE_SHELL_MOUNT_NFS
6 #define CONFIGURE_SHELL_MOUNT_RFS
7 #endif
```

PROGRAMMING INFORMATION:

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

```
1 int rtems_shell_rtems_main_mount(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the mount has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_MOUNT_Command;
```

4.2.22 mv - move files

SYNOPSIS:

```
1 mv [-fiv] source_file target_file
2 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:

```
1 rm -f destination_path && \
2 cp -PRp source_file destination_path && \
3 rm -rf source_file
```

EXIT STATUS:

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

NOTES:

None.

EXAMPLES:

```
1 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:

```
1 int rtems_shell_main_mv(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 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.

4.2.23 `pwd` - print work directory

SYNOPSIS:

```
1 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`:

```
1 SHLL [/] $ pwd  
2 /  
3 SHLL [/] $ cd dev  
4 SHLL [/dev] $ pwd  
5 /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:

```
1 int rtems_shell_rtems_main_pwd(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_PWD_Command;
```

4.2.24 `rmdir` - remove empty directories

SYNOPSIS:

```
1 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 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`:

```
1 SHLL [/] # mkdir joeldir
2 SHLL [/] # rmdir joeldir
3 SHLL [/] # ls joeldir
4 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:

```
1 int rtems_shell_rtems_main_rmdir(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_RMDIR_Command;
```

4.2.25 rm - remove files

SYNOPSIS:

```
1 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:

```
1 SHLL [/] # cp /etc/passwd tmpfile
2 SHLL [/] # cat tmpfile
3 root:*:0:0:root:::/bin/sh
4 rtems:*:1:1:RTEMS Application:::/bin/sh
5 tty:!:2:2:tty owner:::/bin/false
6 SHLL [/] # rm tmpfile
7 SHLL [/] # cat tmpfile
8 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:

```
1 int rtems_shell_main_rm(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the rm has the following prototype: .. code-block:: c

```
extern rtems_shell_cmd_t rtems_shell_RM_Command;
```

4.2.26 umask - set file mode creation mask

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ umask
2 022
3 SHLL [/] $ umask 0666
4 0666
5 SHLL [/] $ umask
6 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:

```
1 int rtems_shell_rtems_main_umask(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the umask has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_UMASK_Command;
```

4.2.27 unmount - unmount disk

SYNOPSIS:

```
1 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:

```
1 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:

```
1 int rtems_shell_rtems_main_unmount(  
2     int    argc,  
3     char **argv  
4 );
```

The configuration structure for the unmount has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_UNMOUNT_Command;
```


MEMORY COMMANDS

5.1 Introduction

The RTEMS shell has the following memory commands:

- *mdump* (page 63) - Display contents of memory
- *wdump* (page 64) - Display contents of memory (word)
- *ldump* (page 65) - Display contents of memory (longword)
- *medit* (page 66) - Modify contents of memory
- *mfill* (page 67) - Fill memory with pattern
- *mmove* (page 68) - Move contents of memory
- *malloc* (page 69) - Obtain information on C Program Heap

5.2 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.

5.2.1 mdump - display contents of memory

SYNOPSIS:

```
1 mdump [address [length [size]]]
```

DESCRIPTION:

This command displays the contents of memory at the address and length in size byte units specified on the command line.

When size is not provided, it defaults to 1 byte units. Values of 1, 2, and 4 are valid; all others will cause an error to be reported.

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:

```

1 SHLL [/] $ mdump 0x10000 32
2 0x0001000000 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3 0x0001001000 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
4 SHLL [/] $ mdump 0x02000000 32
5 0x02000000A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)3.."....!.
6 0x02000010A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 01 .H..)3.."....!.
7 SHLL [/] $ mdump 0x02001000 32
8 0x0200100003 00 80 00 82 10 60 00-81 98 40 00 83 48 00 00 .....`.....H..
9 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:

```

1 int rtems_shell_rtems_main_mdump(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the mdump has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_MDUMP_Command;

```

5.2.2 wdump - display contents of memory (word)**SYNOPSIS:**

```

1 wdump [address [length]]

```

DESCRIPTION:

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

This command is equivalent to `mdump address length 2`.

When length is not provided, it defaults to 320 which is twenty lines of output with eight words 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:

```
1 SHLL [/] $ wdump 0x02010000 32
2 0x02010000 0201 08D8 0201 08C0-0201 08AC 0201 0874 .....t
3 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:

```
1 int rtems_shell_rtems_main_wdump(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the wdump has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_WDUMP_Command;
```

5.2.3 ldump - display contents of memory (longword)

SYNOPSIS:

```
1 ldump [address [length]]
```

DESCRIPTION:

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

This command is equivalent to `mdump address length 4`.

When length is not provided, it defaults to 320 which is twenty lines of output with four longwords 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 ldump:

```

1 SHLL [/] $ ldump 0x02010000 32
2 0x02010000 020108D8 020108C0-020108AC 02010874 .....t
3 0x02010010 020 0894 02010718-02010640 02010798 .....

```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```

1 int rtems_shell_rtems_main_ldump(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the ldump has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_LDUMP_Command;

```

5.2.4 medit - modify contents of memory

SYNOPSIS:

```

1 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:

```

1 SHLL [/] $ mdump 0x02000000 32
2 0x02000000 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)..3.."...!.
3 0x02000010 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 01 .H..)..3.."...!.
4 SHLL [/] $ medit 0x02000000 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09

```

```

5 SHLL [/] $ mdump 0x02000000 32
6 0x02000000 01 02 03 04 05 06 07 08-09 00 22 BC A6 10 21 00 ....."....!.
7 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:

```

1 int rtems_shell_rtems_main_medit(
2     int    argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_MEDIT_Command;

```

5.2.5 mfill - file memory with pattern

SYNOPSIS:

```

1 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.

```

1 SHLL [/] $ mdump 0x23d89a0 16
2 0x023D89A0 A5 A5 A5 A5 A5 A5 A5-FE ED F0 0D 0B AD 0D 06 .....
3 SHLL [/] $ mfill 0x23d89a0 13 0x5a
4 SHLL [/] $ BLOWN STACK!!! Offending task(0x23D4418): id=0x09010001; name=0x0203D908
5 stack covers range 0x23D89A0 - 0x23D99AF (4112 bytes)
6 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:

```

1 int rtems_shell_rtems_main_mfill(
2     int    argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_MFILL_Command;

```

5.2.6 mmove - move contents of memory**SYNOPSIS:**

```

1 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`:

```

1 SHLL [/] $ mdump 0x023d99a0 16
2 0x023D99A0 A5 A5 A5 A5 A5 A5 A5-A5 A5 A5 A5 A5 A5 A5 .....
3 SHLL [/] $ mdump 0x02000000 16
4 0x02000000 A1 48 00 00 29 00 80 33-81 C5 22 BC A6 10 21 00 .H..)3.."....!.
5 SHLL [/] $ mmove 0x023d99a0 0x02000000 13
6 SHLL [/] $ mdump 0x023d99a0 16
7 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:

```
1 int rtems_shell_rtems_main_mmove(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_MMOVE_Command;
```

5.2.7 malloc - obtain information on C program heap

SYNOPSIS:

```
1 malloc [walk]
```

DESCRIPTION:

This command prints information about the current state of the C Program Heap used by the `malloc()` family of calls if no or invalid options are passed to the command. 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
- Size of the allocatable area in bytes
- Minimum free size ever in bytes
- Maximum number of free blocks ever
- Maximum number of blocks searched ever
- Lifetime number of bytes allocated
- Lifetime number of bytes freed
- Total number of searches
- Total number of successful allocations

- Total number of failed allocations
- Total number of successful frees
- Total number of successful resizes

When the subcommand `walk` is specified, then a heap walk will be performed and information about each block is printed out.

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 `malloc` command.

```

1 SHLL [/] $ malloc
2 C Program Heap and RTEMS Workspace are the same.
3 Number of free blocks:                2
4 Largest free block:                  266207504
5 Total bytes free:                    266208392
6 Number of used blocks:               167
7 Largest used block:                  16392
8 Total bytes used:                    83536
9 Size of the allocatable area in bytes: 266291928
10 Minimum free size ever in bytes:    266207360
11 Maximum number of free blocks ever:  6
12 Maximum number of blocks searched ever: 5
13 Lifetime number of bytes allocated:  91760
14 Lifetime number of bytes freed:      8224
15 Total number of searches:            234
16 Total number of successful allocations: 186
17 Total number of failed allocations:   0
18 Total number of successful frees:     19
19 Total number of successful resizes:   0
20 SHLL [/] $ malloc walk
21 malloc walk
22 PASS[0]: page size 8, min block size 48
23 area begin 0x00210210, area end 0xFFFFC000
24 first block 0x00210214, last block 0xFFFFBFDC
25 first free 0x00228084, last free 0x00228354
26 PASS[0]: block 0x00210214: size 88
27 ...
28 PASS[0]: block 0x00220154: size 144
29 PASS[0]: block 0x002201E4: size 168, prev 0x002205BC, next 0x00228354 (= last free)
30 PASS[0]: block 0x0022028C: size 168, prev_size 168
31 ...
32 PASS[0]: block 0x00226E7C: size 4136
33 PASS[0]: block 0x00227EA4: size 408, prev 0x00228084 (= first free), next 0x00226CE4
34 PASS[0]: block 0x0022803C: size 72, prev_size 408
35 PASS[0]: block 0x00228084: size 648, prev 0x0020F75C (= head), next 0x00227EA4
36 PASS[0]: block 0x0022830C: size 72, prev_size 648
37 PASS[0]: block 0x00228354: size 266157192, prev 0x002201E4, next 0x0020F75C (= tail)
38 PASS[0]: block 0xFFFFBFDC: size 4028711480, prev_size 266157192

```

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:

```
1 int rtems_shell_rtems_main_malloc(  
2     int    argc,  
3     char **argv  
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_MALLOC_Command;
```


RTEMS SPECIFIC COMMANDS

6.1 Introduction

The RTEMS shell has the following RTEMS specific commands:

- *shutdown* (page 74) - Shutdown the system
- *cpuuse* (page 74) - print or reset per thread cpu usage
- *stackuse* (page 75) - print per thread stack usage
- *perioduse* (page 76) - print or reset per period usage
- *profreport* (page 78) - print a profiling report
- *wkspc* (page 79) - Display information on Executive Workspace
- *config* (page 80) - Show the system configuration.
- *itask* (page 81) - List init tasks for the system
- *extension* (page 82) - Display information about extensions
- *task* (page 83) - Display information about tasks
- *queue* (page 84) - Display information about message queues
- *sema* (page 85) - display information about semaphores
- *region* (page 86) - display information about regions
- *part* (page 86) - display information about partitions
- *object* (page 87) - Display information about RTEMS objects
- *driver* (page 88) - Display the RTEMS device driver table
- *dname* (page 89) - Displays information about named drivers
- *pthread* (page 90) - Displays information about POSIX threads

6.2 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.

6.2.1 shutdown - Shutdown the system

SYNOPSIS:

```
1 shutdown
```

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 shutdown:

```
1 SHLL [/] $ shutdown
2 System shutting down at user request
```

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_SHUTDOWN` to have this command included.

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

PROGRAMMING INFORMATION:

The configuration structure for the shutdown has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_SHUTDOWN_Command;
```

6.2.2 cpuuse - print or reset per thread cpu usage

SYNOPSIS:

```
1 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`:

```

1 SHLL [/] $ cpuuse
2 CPU Usage by thread
3 ID          NAME          SECONDS  PERCENT
4 0x09010001  IDLE          49.745393  98.953
5 0x0a010001  UI1             0.000000   0.000
6 0x0a010002  SHLL             0.525928   1.046
7 Time since last CPU Usage reset 50.271321 seconds
8 SHLL [/] $ cpuuse -r
9 Resetting CPU Usage information
10 SHLL [/] $ cpuuse
11 CPU Usage by thread
12 ID          NAME          SECONDS  PERCENT
13 0x09010001  IDLE             0.000000   0.000
14 0x0a010001  UI1             0.000000   0.000
15 0x0a010002  SHLL             0.003092  100.000
16 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:

```

1 int rtems_shell_rtems_main_cpuuse(
2     int    argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_CPUUSE_Command;

```

6.2.3 stackuse - print per thread stack usage

SYNOPSIS:

```

1 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`:

```

1 SHLL [/] $ stackuse
2 Stack usage by thread
3 ID      NAME    LOW          HIGH        CURRENT     AVAILABLE    USED
4 0x09010001 IDLE 0x023d89a0 - 0x023d99af 0x023d9760    4096        608
5 0x0a010001 UI1  0x023d9f30 - 0x023daf3f 0x023dad18    4096       1804
6 0x0a010002 SHLL 0x023db4c0 - 0x023df4cf 0x023de9d0   16384       5116
7 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:

```

1 int rtems_shell_rtems_main_stackuse(
2     int    argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_STACKUSE_Command;

```

6.2.4 perioduse - print or reset per period usage**SYNOPSIS:**

```

1 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:

```

1 SHLL [/] $ perioduse
2 Period information by period
3 --- CPU times are in seconds ---
4 --- Wall times are in seconds ---
5 ID      OWNER COUNT MISSED      CPU TIME      WALL TIME
6 MIN/MAX/AVG      MIN/MAX/AVG
7 0x42010001 TA1    502      0 0:000039/0:042650/0:004158 0:000039/0:020118/0:002848
8 0x42010002 TA2    502      0 0:000041/0:042657/0:004309 0:000041/0:020116/0:002848
9 0x42010003 TA3    501      0 0:000041/0:041564/0:003653 0:000041/0:020003/0:002814
10 0x42010004 TA4    501      0 0:000043/0:044075/0:004911 0:000043/0:020004/0:002814
11 0x42010005 TA5     10      0 0:000065/0:005413/0:002739 0:000065/1:000457/0:041058
12 MIN/MAX/AVG      MIN/MAX/AVG
13 SHLL [/] $ perioduse -r
14 Resetting Period Usage information
15 SHLL [/] $ perioduse
16 --- CPU times are in seconds ---
17 --- Wall times are in seconds ---
18 ID      OWNER COUNT MISSED      CPU TIME      WALL TIME
19 MIN/MAX/AVG      MIN/MAX/AVG
20 0x42010001 TA1      0      0
21 0x42010002 TA2      0      0
22 0x42010003 TA3      0      0
23 0x42010004 TA4      0      0
24 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:

```

1 int rtems_shell_rtems_main_perioduse(
2     int  argc,
3     char **argv
4 );

```

The configuration structure for the perioduse has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_PERIODUSE_Command;
```

6.2.5 profreport - print a profiling report

SYNOPSIS:

```
1 profreport
```

DESCRIPTION:

This command may be used to print a profiling report if profiling is built into the RTEMS kernel.

EXIT STATUS:

This command returns 0.

NOTES:

Profiling must be enabled at build configuration time to get profiling information.

EXAMPLES:

The following is an example of how to use profreport:

```
1 SHLL [/] $ profreport
2 <ProfilingReport name="Shell">
3 <PerCPUProfilingReport processorIndex="0">
4 <MaxThreadDispatchDisabledTime unit="ns">10447</MaxThreadDispatchDisabledTime>
5 <MeanThreadDispatchDisabledTime unit="ns">2</MeanThreadDispatchDisabledTime>
6 <TotalThreadDispatchDisabledTime unit="ns">195926627</TotalThreadDispatchDisabledTime>
7 <ThreadDispatchDisabledCount>77908688</ThreadDispatchDisabledCount>
8 <MaxInterruptDelay unit="ns">0</MaxInterruptDelay>
9 <MaxInterruptTime unit="ns">688</MaxInterruptTime>
10 <MeanInterruptTime unit="ns">127</MeanInterruptTime>
11 <TotalInterruptTime unit="ns">282651157</TotalInterruptTime>
12 <InterruptCount>2215855</InterruptCount>
13 </PerCPUProfilingReport>
14 <PerCPUProfilingReport processorIndex="1">
15 <MaxThreadDispatchDisabledTime unit="ns">9053</MaxThreadDispatchDisabledTime>
16 <MeanThreadDispatchDisabledTime unit="ns">41</MeanThreadDispatchDisabledTime>
17 <TotalThreadDispatchDisabledTime unit="ns">3053830335</TotalThreadDispatchDisabledTime>
18 <ThreadDispatchDisabledCount>73334202</ThreadDispatchDisabledCount>
19 <MaxInterruptDelay unit="ns">0</MaxInterruptDelay>
20 <MaxInterruptTime unit="ns">57</MaxInterruptTime>
21 <MeanInterruptTime unit="ns">35</MeanInterruptTime>
22 <TotalInterruptTime unit="ns">76980203</TotalInterruptTime>
23 <InterruptCount>2141179</InterruptCount>
24 </PerCPUProfilingReport>
25 <SMPLockProfilingReport name="SMP lock stats">
26 <MaxAcquireTime unit="ns">608</MaxAcquireTime>
27 <MaxSectionTime unit="ns">1387</MaxSectionTime>
28 <MeanAcquireTime unit="ns">112</MeanAcquireTime>
29 <MeanSectionTime unit="ns">338</MeanSectionTime>
30 <TotalAcquireTime unit="ns">119031</TotalAcquireTime>
31 <TotalSectionTime unit="ns">357222</TotalSectionTime>
32 <UsageCount>1055</UsageCount>
33 <ContentionCount initialQueueLength="0">1055</ContentionCount>
```

```

34 <ContentionCount initialQueueLength="1">0</ContentionCount>
35 <ContentionCount initialQueueLength="2">0</ContentionCount>
36 <ContentionCount initialQueueLength="3">0</ContentionCount>
37 </SMPLockProfilingReport>
38 <SMPLockProfilingReport name="Giant">
39 <MaxAcquireTime unit="ns">4186</MaxAcquireTime>
40 <MaxSectionTime unit="ns">7575</MaxSectionTime>
41 <MeanAcquireTime unit="ns">160</MeanAcquireTime>
42 <MeanSectionTime unit="ns">183</MeanSectionTime>
43 <TotalAcquireTime unit="ns">1772793111</TotalAcquireTime>
44 <TotalSectionTime unit="ns">2029733879</TotalSectionTime>
45 <UsageCount>11039140</UsageCount>
46 <ContentionCount initialQueueLength="0">11037655</ContentionCount>
47 <ContentionCount initialQueueLength="1">1485</ContentionCount>
48 <ContentionCount initialQueueLength="2">0</ContentionCount>
49 <ContentionCount initialQueueLength="3">0</ContentionCount>
50 </SMPLockProfilingReport>
51 </ProfilingReport>

```

CONFIGURATION:

When building a custom command set, define `CONFIGURE_SHELL_COMMAND_PROFREPORT` to have this command included.

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

PROGRAMMING INFORMATION:

The configuration structure for the profreport has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_PROFREPORT_Command;

```

6.2.6 wkspc - display information on executive workspace**SYNOPSIS:**

```

1 wkspc

```

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`:

```

1 SHLL [/] $ wkspace
2 Number of free blocks: 1
3 Largest free block:    132336
4 Total bytes free:      132336
5 Number of used blocks: 36
6 Largest used block:    16408
7 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:

```

1 int rtems_shell_rtems_main_wkspace(
2     int    argc,
3     char **argv
4 );

```

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

```

1 extern rtems_shell_cmd_t rtems_shell_WKSPACE_Command;

```

6.2.7 config - show the system configuration.**SYNOPSIS:**

```

1 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`:

```

1 SHLL [/] $ config
2 INITIAL (startup) Configuration Info
3
4 WORKSPACE      start: 0x23d22e0; size: 0x2dd20
5 TIME           usec/tick: 10000; tick/timeslice: 50; tick/sec: 100
6 MAXIMUMS       tasks: 20; timers: 0; sems: 50; que's: 20; ext's: 1
7 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:

```

1 int rtems_shell_rtems_main_config(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the config has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_CONFIG_Command;

```

6.2.8 itask - list init tasks for the system

SYNOPSIS:

```

1 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`:

```

1 SHLL [/] $ itask
2 #   NAME   ENTRY      ARGUMENT      PRIO   MODES  ATTRIBUTES  STACK SIZE
3 -----
4 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:

```
1 int rtems_shell_rtems_main_itask(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_ITASK_Command;
```

6.2.9 extension - display information about extensions**SYNOPSIS:**

```
1 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.

```
1 SHLL [/] $ extension
2 ID      NAME
3 -----
```

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:

```
1 int rtems_shell_rtems_main_extension(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the extension has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_EXTENSION_Command;
```

6.2.10 task - display information about tasks

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ task
2 ID      NAME  PRIO  STAT  MODES  EVENTS  WAITID  WAITARG  NOTES
3 -----
4 0a010001  UI1      1    SUSP  P:T:nA  NONE
5 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:

```

1 int rtems_shell_rtems_main_task(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the task has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_TASK_Command;

```

6.2.11 queue - display information about message queues

SYNOPSIS:

```

1 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.

```

1 SHLL [/] $ queue
2 ID      NAME  ATTRIBUTES  PEND  MAXPEND  MAXSIZE
3 -----

```

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:

```

1 int rtems_shell_rtems_main_queue(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the queue has the following prototype:


```
1 extern rtems_shell_cmd_t rtems_shell_QUEUE_Command;
```

6.2.12 sema - display information about semaphores

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ sema
2 ID      NAME  ATTR      PRICEIL CURR_CNT HOLDID
3 -----
4 1a010001  LBI0   PR:BI:IN    0        1    00000000
5 1a010002  TRmi   PR:BI:IN    0        1    00000000
6 1a010003  LBI00   PR:BI:IN    0        1    00000000
7 1a010004  TRia   PR:BI:IN    0        1    00000000
8 1a010005  TRoa   PR:BI:IN    0        1    00000000
9 1a010006  TRxa   <assoc.c: BAD NAME> 0    0 09010001
10 1a010007  LBI01   PR:BI:IN    0        1    00000000
11 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:

```
1 int rtems_shell_rtems_main_sema(
2     int  argc,
3     char **argv
4 );
```

The configuration structure for the sema has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_SEMA_Command;
```

6.2.13 region - display information about regions

SYNOPSIS:

```
1 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.

```
1 SHLL [/] $ region
2 ID      NAME  ATTR      STARTADDR LENGTH  PAGE_SIZE USED_BLOCKS
3 -----
```

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:

```
1 int rtems_shell_rtems_main_region(
2     int  argc,
3     char **argv
4 );
```

The configuration structure for the region has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_REGION_Command;
```

6.2.14 part - display information about partitions

SYNOPSIS:

```
1 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.

```
1 SHLL [/] $ part
2 ID      NAME  ATTR      STARTADDR LENGTH  BUF_SIZE  USED_BLOCKS
3 -----
```

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:

```
1 int rtems_shell_rtems_main_part(
2     int  argc,
3     char **argv
4 );
```

The configuration structure for the part has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_PART_Command;
```

6.2.15 object - display information about RTEMS objects**SYNOPSIS:**

```
1 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:

```

1 SHLL [/] $ object 0a010001 1a010002
2 ID      NAME  PRIO  STAT  MODES  EVENTS  WAITID  WAITARG  NOTES
3 -----
4 0a010001  UI1    1   SUSP  P:T:nA  NONE
5 ID      NAME  ATTR          PRICEIL  CURR_CNT  HOLDID
6 -----
7 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:

```

1 int rtems_shell_rtems_main_object(
2     int  argc,
3     char **argv
4 );

```

The configuration structure for the object has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_OBJECT_Command;

```

6.2.16 driver - display the RTEMS device driver table

SYNOPSIS:

```

1 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:

```

1 SHLL [/] $ driver
2 Major      Entry points
3 -----
4 0          init: [0x200256c]; control: [0x20024c8]
5 open: [0x2002518]; close: [0x2002504]
6 read: [0x20024f0]; write: [0x20024dc]
7 1          init: [0x20023fc]; control: [0x2002448]
8 open: [0x0]; close: [0x0]
9 read: [0x0]; write: [0x0]
10 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:

```

1 int rtems_shell_rtems_main_driver(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the driver has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_DRIVER_Command;

```

6.2.17 dname - displays information about named drivers

SYNOPSIS:

```

1 dname

```

DESCRIPTION:

WARNING! 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:

```
1 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:

```
1 int rtems_shell_rtems_main_dname(
2     int    argc,
3     char **argv
4 );
```

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

```
1 extern rtems_shell_cmd_t rtems_shell_DNAME_Command;
```

6.2.18 pthread - display information about POSIX threads

SYNOPSIS:

```
1 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:

```
1 SHLL [/] $ pthread
2 ID      NAME      PRI  STATE  MODES  EVENTS  WAITID  WAITARG  NOTES
3 -----
4 0b010002  Main             133  READY  P:T:nA  NONE    43010001 0x7b1148
5 0b010003  ISR              133  Wcvar  P:T:nA  NONE    43010003 0x7b1148
6 0b01000c              133  READY  P:T:nA  NONE    33010002 0x7b1148
7 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.

NETWORK COMMANDS

7.1 Introduction

The RTEMS shell has the following network commands:

- *netstats* (page 93) - obtain network statistics
- *ifconfig* (page 96) - configure a network interface
- *route* (page 97) - show or manipulate the IP routing table
- *ping* (page 98) - ping a host or IP address

7.2 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.

7.2.1 netstats - obtain network statistics

SYNOPSIS:

```
1 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 using the netstats command to print the IP routing table:

```
1 [/] $ netstats -i
2 Destination      Gateway/Mask/Hw    Flags    Refs    Use Expire Interface
3 default          192.168.1.14      UGS      0       0       0 eth1
4 192.168.1.0      255.255.255.0     U        0       0       1 eth1
5 192.168.1.14     00:A0:C8:1C:EE:28 UHL      1       0     1219 eth1
6 192.168.1.51     00:1D:7E:0C:D0:7C UHL      0     840     1202 eth1
7 192.168.1.151    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:

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

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

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

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

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

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

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

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

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

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

```
1 [/] $ netstats -t
2 ***** TCP Statistics *****
3 connections accepted        1
4 connections established     1
5 segs where we tried to get rtt      34
6 times we succeeded           35
7 delayed acks sent           2
8 total packets sent           37
9 data packets sent            35
10 data bytes sent             2618
11 ack-only packets sent       2
12 total packets received      47
13 packets received in sequence    12
14 bytes received in sequence    307
15 rcvd ack packets            35
16 bytes acked by rcvd acks      2590
17 times hdr predict ok for acks    27
18 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:

```
1 int rtems_shell_rtems_main_netstats(
2     int  argc,
3     char **argv
4 );
```

The configuration structure for the netstats has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_NETSTATS_Command;
```

7.2.2 ifconfig - configure a network interface

SYNOPSIS:

```

1 ifconfig
2 ifconfig interface
3 ifconfig interface \[up|down]
4 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:

```

1 ***** INTERFACE STATISTICS *****
2 ***** eth1 *****
3 Ethernet Address: 00:04:9F:00:5B:21
4 Address:192.168.1.244   Broadcast Address:192.168.1.255   Net mask:255.255.255.0
5 Flags: Up Broadcast Running Active Multicast
6 Send queue limit:50   length:1   Dropped:0
7 Rx Interrupts:5391           Not First:0           Not Last:0
8 Giant:0                 Non-octet:0
9 Bad CRC:0               Overrun:0               Collision:0
10 Tx Interrupts:5256        Deferred:0           Late Collision:0
11 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:

```

1 int rtems_shell_rtems_main_ifconfig(
2     int    argc,
3     char **argv
4 );

```

The configuration structure for the ifconfig has the following prototype:

```

1 extern rtems_shell_cmd_t rtems_shell_IFCONFIG_Command;

```

7.2.3 route - show or manipulate the ip routing table

SYNOPSIS:

```
1 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:

```
1 route [add|del] -net IP_ADDRESS gw GATEWAY_ADDRESS [netmask MASK]
2 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:

```
1 [/] $ route
2 Destination      Gateway/Mask/Hw    Flags    Refs      Use Expire Interface
3 default          192.168.1.14       UGS      0          0      0 eth1
4 192.168.1.0       255.255.255.0      U        0          0      1 eth1
5 192.168.1.14      00:A0:C8:1C:EE:28  UHL      1          0    1444 eth1
6 192.168.1.51      00:1D:7E:0C:D0:7C  UHL      0    10844    1202 eth1
7 192.168.1.151     00:1C:23:B2:0F:BB  UHL      2         37    1399 eth1
8 [/] $ route add -net 192.168.3.0 gw 192.168.1.14
9 [/] $ route
10 Destination      Gateway/Mask/Hw    Flags    Refs      Use Expire Interface
11 default          192.168.1.14       UGS      0          0      0 eth1
12 192.168.1.0       255.255.255.0      U        0          0      1 eth1
13 192.168.1.14      00:A0:C8:1C:EE:28  UHL      2          0    1498 eth1
14 192.168.1.51      00:1D:7E:0C:D0:7C  UHL      0    14937    1202 eth1
15 192.168.1.151     00:1C:23:B2:0F:BB  UHL      2         96    1399 eth1
16 192.168.3.0       192.168.1.14       UGS      0          0      0 eth1
17 [/] $ route del -net 192.168.3.0 gw 192.168.1.14
18 [/] $ route
19 Destination      Gateway/Mask/Hw    Flags    Refs      Use Expire Interface
20 default          192.168.1.14       UGS      0          0      0 eth1
21 192.168.1.0       255.255.255.0      U        0          0      1 eth1
22 192.168.1.14      00:A0:C8:1C:EE:28  UHL      1          0    1498 eth1
23 192.168.1.51      00:1D:7E:0C:D0:7C  UHL      0    15945    1202 eth1
24 192.168.1.151     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:

```
1 int rtems_shell_rtems_main_route(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the route has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_ROUTE_Command;
```

7.2.4 ping - ping a host or IP address

SYNOPSIS:

```
1 ping [-AaDdfnoQqRrv] [-c count] [-G sweepmaxsize] [-g sweepminsize]
2 [-h sweepincrsz] [-i wait] [-l preload] [-M mask | time] [-m ttl]
3 [-p pattern] [-S src_addr] [-s packetsize] [-t timeout]
4 [-W waittime] [-z tos] host
5 ping [-AaDdfLnoQqRrv] [-c count] [-I iface] [-i wait] [-l preload]
6 [-M mask | time] [-m ttl] [-p pattern] [-S src_addr]
7 [-s packetsize] [-T ttl] [-t timeout] [-W waittime]
8 [-z tos] mcast-group
```

DESCRIPTION:

The ping utility uses the ICMP protocol's mandatory `ECHO_REQUEST` datagram to elicit an ICMP `ECHO_RESPONSE` from a host or gateway. `ECHO_REQUEST` datagrams ("pings") have an IP and ICMP header, followed by a "struct timeval" and then an arbitrary number of "pad" bytes used to fill out the packet. The options are as follows:

- A** Audible. Output a bell (ASCII 0x07) character when no packet is received before the next packet is transmitted. To cater for round-trip times that are longer than the interval between transmissions, further missing packets cause a bell only if the maximum number of unreceived packets has increased.
- a** Audible. Include a bell (ASCII 0x07) character in the output when any packet is received. This option is ignored if other format options are present.
- c count** Stop after sending (and receiving) count `ECHO_RESPONSE` packets. If this option is not specified, ping will operate until interrupted. If this option is specified in conjunction with ping sweeps, each sweep will consist of count packets.
- D** Set the Don't Fragment bit.
- d** Set the `SO_DEBUG` option on the socket being used.
- f** Flood ping. Outputs packets as fast as they come back or one hundred times per second, whichever is more. For every `ECHO_REQUEST` sent a period "." is printed, while for

every ECHO_REPLY received a backspace is printed. This provides a rapid display of how many packets are being dropped. Only the super-user may use this option. This can be very hard on a network and should be used with caution.

- G *sweepmaxsize*** Specify the maximum size of ICMP payload when sending sweeping pings. This option is required for ping sweeps.
- g *sweepminsize*** Specify the size of ICMP payload to start with when sending sweeping pings. The default value is 0.
- h *sweepincrsz*** Specify the number of bytes to increment the size of ICMP payload after each sweep when sending sweeping pings. The default value is 1.
- I *iface*** Source multicast packets with the given interface address. This flag only applies if the ping destination is a multicast address.
- i *wait*** Wait *wait* seconds between sending each packet. The default is to wait for one second between each packet. The wait time may be fractional, but only the super-user may specify values less than 1 second. This option is incompatible with the -f option.
- L** Suppress loopback of multicast packets. This flag only applies if the ping destination is a multicast address.
- l *preload*** If *preload* is specified, ping sends that many packets as fast as possible before falling into its normal mode of behavior. Only the super-user may use this option.
- M *mask* | *time*** Use ICMP_MASKREQ or ICMP_TSTAMP instead of ICMP_ECHO. For *mask*, print the netmask of the remote machine. Set the net.inet.icmp.maskrepl MIB variable to enable ICMP_MASKREPLY. For *time*, print the origination, reception and transmission timestamps.
- m *tll*** Set the IP Time To Live for outgoing packets. If not specified, the kernel uses the value of the net.inet.ip.ttl MIB variable.
- n** Numeric output only. No attempt will be made to lookup symbolic names for host addresses.
- o** Exit successfully after receiving one reply packet.
- p *pattern*** You may specify up to 16 “pad” bytes to fill out the packet you send. This is useful for diagnosing data-dependent problems in a network. For example, “-p ff” will cause the sent packet to be filled with all ones.
- Q** Somewhat quiet output. Don’t display ICMP error messages that are in response to our query messages. Originally, the -v flag was required to display such errors, but -v displays all ICMP error messages. On a busy machine, this output can be overbearing. Without the -Q flag, ping prints out any ICMP error messages caused by its own ECHO_REQUEST messages.
- q** Quiet output. Nothing is displayed except the summary lines at startup time and when finished.
- R** Record route. Includes the RECORD_ROUTE option in the ECHO_REQUEST packet and displays the route buffer on returned packets. Note that the IP header is only large enough for nine such routes; the traceroute(8) command is usually better at determining the route packets take to a particular destination. If more routes come back than should, such as due to an illegal spoofed packet, ping will print the route list and then truncate it at the correct spot. Many hosts ignore or discard the RECORD_ROUTE option.

- r** Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped).
- S *src_addr*** Use the following IP address as the source address in outgoing packets. On hosts with more than one IP address, this option can be used to force the source address to be something other than the IP address of the interface the probe packet is sent on. If the IP address is not one of this machine's interface addresses, an error is returned and nothing is sent.
- s *packetsize*** Specify the number of data bytes to be sent. The default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data. Only the super-user may specify values more than default. This option cannot be used with ping sweeps.
- T *tll*** Set the IP Time To Live for multicasted packets. This flag only applies if the ping destination is a multicast address.
- t *timeout*** Specify a timeout, in seconds, before ping exits regardless of how many packets have been received.
- v** Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.
- W *waittime*** Time in milliseconds to wait for a reply for each packet sent. If a reply arrives later, the packet is not printed as replied, but considered as replied when calculating statistics.
- z *tos*** Use the specified type of service.

EXIT STATUS:

The ping utility exits with one of the following values:

0 At least one response was heard from the specified host.

2 The transmission was successful but no responses were received.

any other value an error occurred. These values are defined in <sysexits.h>.

NOTES:

When using ping for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the round-trip time statistics. When the specified number of packets have been sent a brief summary is displayed, showing the number of packets sent and received, and the minimum, mean, maximum, and standard deviation of the round-trip times.

This program is intended for use in network testing, measurement and management. Because of the load it can impose on the network, it is unwise to use ping during normal operations or from automated scripts.

This command can fail if more than the FD_SET size number of file descriptors are open.

EXAMPLES:

The following is an example of how to use ping to ping:


```
1 [/] # ping 10.10.10.1
2 PING 10.10.10.1 (10.10.10.1): 56 data bytes
3 64 bytes from 10.10.10.1: icmp_seq=0 ttl=63 time=0.356 ms
4 64 bytes from 10.10.10.1: icmp_seq=1 ttl=63 time=0.229 ms
5 64 bytes from 10.10.10.1: icmp_seq=2 ttl=63 time=0.233 ms
6 64 bytes from 10.10.10.1: icmp_seq=3 ttl=63 time=0.235 ms
7 64 bytes from 10.10.10.1: icmp_seq=4 ttl=63 time=0.229 ms
8 --- 10.10.10.1 ping statistics ---
9 5 packets transmitted, 5 packets received, 0.0% packet loss
10 round-trip min/avg/max/stddev = 0.229/0.256/0.356/0.050 ms
11 [/] # ping -f -c 10000 10.10.10.1
12 PING 10.10.10.1 (10.10.10.1): 56 data bytes
13 .
14 --- 10.10.10.1 ping statistics ---
15 10000 packets transmitted, 10000 packets received, 0.0% packet loss
16 round-trip min/avg/max/stddev = 0.154/0.225/0.533/0.027 ms
```

CONFIGURATION:

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

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

PROGRAMMING INFORMATION:

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

```
1 int rtems_shell_rtems_main_ping(
2     int    argc,
3     char **argv
4 );
```

The configuration structure for the ping has the following prototype:

```
1 extern rtems_shell_cmd_t rtems_shell_PING_Command;
```


FUNCTION AND VARIABLE INDEX

CONCEPT INDEX

- [genindex](#)
- [search](#)

INDEX

- alias, 13
- blksync, 28
- cat, 29
- cd, 29
- chdir, 30
- chmod, 31
- chroot, 33
- cmdchmod, 15
- cmdchown, 14
- cmdls, 14
- config, 80
- CONFIGURE_SHELL_COMMAND_ALIAS, 13
- CONFIGURE_SHELL_COMMAND_BLKSYNC, 28
- CONFIGURE_SHELL_COMMAND_CAT, 29
- CONFIGURE_SHELL_COMMAND_CD, 30
- CONFIGURE_SHELL_COMMAND_CHDIR, 31
- CONFIGURE_SHELL_COMMAND_CHMOD, 32
- CONFIGURE_SHELL_COMMAND_CHROOT, 33
- CONFIGURE_SHELL_COMMAND_CMDCHMOD, 15
- CONFIGURE_SHELL_COMMAND_CMDCHOWN, 15
- CONFIGURE_SHELL_COMMAND_CMDLS, 14
- CONFIGURE_SHELL_COMMAND_CONFIG, 81
- CONFIGURE_SHELL_COMMAND_CP, 36
- CONFIGURE_SHELL_COMMAND_CPUUSE, 75
- CONFIGURE_SHELL_COMMAND_DATE, 16
- CONFIGURE_SHELL_COMMAND_DD, 39
- CONFIGURE_SHELL_COMMAND_DEBUGRFS, 40
- CONFIGURE_SHELL_COMMAND_DF, 41
- CONFIGURE_SHELL_COMMAND_DIR, 42
- CONFIGURE_SHELL_COMMAND_DNAME, 90
- CONFIGURE_SHELL_COMMAND_DRIVER, 89
- CONFIGURE_SHELL_COMMAND_ECHO, 18
- CONFIGURE_SHELL_COMMAND_EXTENSION, 82
- CONFIGURE_SHELL_COMMAND_FDISK, 43
- CONFIGURE_SHELL_COMMAND_GETENV, 22
- CONFIGURE_SHELL_COMMAND_HEXDUMP, 45
- CONFIGURE_SHELL_COMMAND_ID, 19
- CONFIGURE_SHELL_COMMAND_IFCONFIG, 96
- CONFIGURE_SHELL_COMMAND_ITASK, 82
- CONFIGURE_SHELL_COMMAND_LDUMP, 66
- CONFIGURE_SHELL_COMMAND_LN, 47
- CONFIGURE_SHELL_COMMAND_LOGOFF, 25
- CONFIGURE_SHELL_COMMAND_LS, 48
- CONFIGURE_SHELL_COMMAND_MALLOC, 71
- CONFIGURE_SHELL_COMMAND_MD5, 49
- CONFIGURE_SHELL_COMMAND_MDUMP, 64
- CONFIGURE_SHELL_COMMAND_MEDIT, 67
- CONFIGURE_SHELL_COMMAND_MFILL, 68
- CONFIGURE_SHELL_COMMAND_MKDIR, 50
- CONFIGURE_SHELL_COMMAND_MKDOS, 51
- CONFIGURE_SHELL_COMMAND_MKNOD, 52
- CONFIGURE_SHELL_COMMAND_MKRFS, 53
- CONFIGURE_SHELL_COMMAND_MMOVE, 69
- CONFIGURE_SHELL_COMMAND_MOUNT, 55
- CONFIGURE_SHELL_COMMAND_MV, 56
- CONFIGURE_SHELL_COMMAND_NETSTATS, 95
- CONFIGURE_SHELL_COMMAND_OBJECT, 88
- CONFIGURE_SHELL_COMMAND_PART, 87
- CONFIGURE_SHELL_COMMAND_PERIODUSE, 77
- CONFIGURE_SHELL_COMMAND_PING, 101

CONFIGURE_SHELL_COMMAND_	CONFIGURE_SHELL_NO_COMMAND_
PROFREPORT, 79	CONFIG, 81
CONFIGURE_SHELL_COMMAND_PWD, 57	CONFIGURE_SHELL_NO_COMMAND_CP, 36
CONFIGURE_SHELL_COMMAND_QUEUE, 84	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_REGION,	CPUUSE, 75
86	CONFIGURE_SHELL_NO_COMMAND_DATE,
CONFIGURE_SHELL_COMMAND_RM, 59	16
CONFIGURE_SHELL_COMMAND_RMDIR, 58	CONFIGURE_SHELL_NO_COMMAND_DD, 39
CONFIGURE_SHELL_COMMAND_ROUTE, 97	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_RTC, 25	DEBUGRFS, 40
CONFIGURE_SHELL_COMMAND_SEMA, 85	CONFIGURE_SHELL_NO_COMMAND_DF, 41
CONFIGURE_SHELL_COMMAND_SETENV,	CONFIGURE_SHELL_NO_COMMAND_DIR,
23	42
CONFIGURE_SHELL_COMMAND_	CONFIGURE_SHELL_NO_COMMAND_
SHUTDOWN, 74	DNAME, 90
CONFIGURE_SHELL_COMMAND_SLEEP, 19	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_STACKUSE,	DRIVER, 89
76	CONFIGURE_SHELL_NO_COMMAND_ECHO,
CONFIGURE_SHELL_COMMAND_TASK, 83	18
CONFIGURE_SHELL_COMMAND_TIME, 24	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_TTY, 20	EXTENSION, 82
CONFIGURE_SHELL_COMMAND_UMASK, 60	CONFIGURE_SHELL_NO_COMMAND_FDISK,
CONFIGURE_SHELL_COMMAND_	43
UNMOUNT, 61	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_	GETENV, 22
UNSETENV, 23	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_COMMAND_WDUMP,	HEXDUMP, 45
65	CONFIGURE_SHELL_NO_COMMAND_ID, 19
CONFIGURE_SHELL_COMMAND_WHOAMI,	CONFIGURE_SHELL_NO_COMMAND_
21	IFCONFIG, 96
CONFIGURE_SHELL_COMMAND_WKSPACE,	CONFIGURE_SHELL_NO_COMMAND_ITASK,
80	82
CONFIGURE_SHELL_NO_COMMAND_ALIAS,	CONFIGURE_SHELL_NO_COMMAND_
13	LDUMP, 66
CONFIGURE_SHELL_NO_COMMAND_	CONFIGURE_SHELL_NO_COMMAND_LN, 47
BLKSYNC, 28	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_NO_COMMAND_CAT,	LOGOFF, 25
29	CONFIGURE_SHELL_NO_COMMAND_LS, 48
CONFIGURE_SHELL_NO_COMMAND_CD, 30	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_NO_COMMAND_	MALLOC, 71
CHDIR, 31	CONFIGURE_SHELL_NO_COMMAND_MD5,
CONFIGURE_SHELL_NO_COMMAND_	49
CHMOD, 32	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_NO_COMMAND_	MDUMP, 64
CHROOT, 33	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_NO_COMMAND_	MEDIT, 67
CMDCHMOD, 15	CONFIGURE_SHELL_NO_COMMAND_MFILL,
CONFIGURE_SHELL_NO_COMMAND_	68
CMDCHOWN, 15	CONFIGURE_SHELL_NO_COMMAND_
CONFIGURE_SHELL_NO_COMMAND_	MKDIR, 50
CMDLS, 14	CONFIGURE_SHELL_NO_COMMAND_

MKDOS, 51	CONFIGURE_SHELL_NO_COMMAND_UMASK, 60
CONFIGURE_SHELL_NO_COMMAND_MKNOD, 52	CONFIGURE_SHELL_NO_COMMAND_UNMOUNT, 61
CONFIGURE_SHELL_NO_COMMAND_MKRFS, 53	CONFIGURE_SHELL_NO_COMMAND_UNSETENV, 23
CONFIGURE_SHELL_NO_COMMAND_MMOVE, 69	CONFIGURE_SHELL_NO_COMMAND_WDUMP, 65
CONFIGURE_SHELL_NO_COMMAND_MOUNT, 55	CONFIGURE_SHELL_NO_COMMAND_WHOAMI, 21
CONFIGURE_SHELL_NO_COMMAND_MV, 56	CONFIGURE_SHELL_NO_COMMAND_WKSPACE, 80
CONFIGURE_SHELL_NO_COMMAND_NETSTATS, 95	cp, 34
CONFIGURE_SHELL_NO_COMMAND_OBJECT, 88	cpuuse, 74
CONFIGURE_SHELL_NO_COMMAND_PART, 87	crypt_add_format, 9
CONFIGURE_SHELL_NO_COMMAND_PERIODUSE, 77	date, 16
CONFIGURE_SHELL_NO_COMMAND_PING, 101	dd, 36
CONFIGURE_SHELL_NO_COMMAND_PROFREPORT, 79	debugrfs, 39
CONFIGURE_SHELL_NO_COMMAND_PWD, 57	df, 41
CONFIGURE_SHELL_NO_COMMAND_QUEUE, 84	dir, 42
CONFIGURE_SHELL_NO_COMMAND_REGION, 86	dname, 89
CONFIGURE_SHELL_NO_COMMAND_RM, 59	driver, 88
CONFIGURE_SHELL_NO_COMMAND_RMDIR, 58	echo, 17
CONFIGURE_SHELL_NO_COMMAND_ROUTE, 97	exit, 26
CONFIGURE_SHELL_NO_COMMAND_RTC, 25	extension, 82
CONFIGURE_SHELL_NO_COMMAND_SEMA, 85	fdisk, 42
CONFIGURE_SHELL_NO_COMMAND_SETENV, 23	getenv, 21
CONFIGURE_SHELL_NO_COMMAND_SHUTDOWN, 74	help, 12
CONFIGURE_SHELL_NO_COMMAND_SLEEP, 19	hexdump, 43
CONFIGURE_SHELL_NO_COMMAND_STACKUSE, 76	id, 19
CONFIGURE_SHELL_NO_COMMAND_TASK, 83	ifconfig, 96
CONFIGURE_SHELL_NO_COMMAND_TIME, 24	initialization, 9, 10
CONFIGURE_SHELL_NO_COMMAND_TTY, 20	itask, 81
	ldump, 65
	ln, 46
	logoff, 25
	ls, 47
	malloc, 69
	md5, 48
	mdump, 63
	medit, 66
	mfill, 67
	mkdir, 49
	mkdos, 50
	mknod, 51
	mkrfns, 53

mmove, 68
mount, 54
mv, 56

netstats, 93

object, 87

part, 86
perioduse, 76
ping, 98
profreport, 78
pthread, 90
pwd, 57

queue, 84

region, 86
rm, 59
rmdir, 58
route, 97
rtc, 25
rtems_shell_init, 9
rtems_shell_login_check, 10
rtems_shell_main_cp, 36
rtems_shell_main_mv, 57
rtems_shell_main_rm, 59
rtems_shell_rtems_main_alias, 13
rtems_shell_rtems_main_blksync, 28
rtems_shell_rtems_main_cat, 29
rtems_shell_rtems_main_cd, 30
rtems_shell_rtems_main_chdir, 31
rtems_shell_rtems_main_chmod, 32
rtems_shell_rtems_main_chroot, 33
rtems_shell_rtems_main_config, 81
rtems_shell_rtems_main_cpuuse, 75
rtems_shell_rtems_main_date, 16
rtems_shell_rtems_main_dd, 39
rtems_shell_rtems_main_debugrfs, 40
rtems_shell_rtems_main_df, 41
rtems_shell_rtems_main_dir, 42
rtems_shell_rtems_main_dname, 90
rtems_shell_rtems_main_driver, 89
rtems_shell_rtems_main_echo, 18
rtems_shell_rtems_main_extension, 83
rtems_shell_rtems_main_getenv, 22
rtems_shell_rtems_main_hexdump, 46
rtems_shell_rtems_main_id, 20
rtems_shell_rtems_main_ifconfig, 96
rtems_shell_rtems_main_itask, 82
rtems_shell_rtems_main_ldump, 66
rtems_shell_rtems_main_ln, 47
rtems_shell_rtems_main_logoff, 25

rtems_shell_rtems_main_ls, 48
rtems_shell_rtems_main_malloc, 71
rtems_shell_rtems_main_md5, 49
rtems_shell_rtems_main_mdump, 64
rtems_shell_rtems_main_medit, 67
rtems_shell_rtems_main_mfill, 68
rtems_shell_rtems_main_mkdir, 50
rtems_shell_rtems_main_mkdos, 51
rtems_shell_rtems_main_mknod, 52
rtems_shell_rtems_main_mkrfs, 54
rtems_shell_rtems_main_mmove, 69
rtems_shell_rtems_main_mount, 55
rtems_shell_rtems_main_netstats, 95
rtems_shell_rtems_main_object, 88
rtems_shell_rtems_main_part, 87
rtems_shell_rtems_main_perioduse, 77
rtems_shell_rtems_main_ping, 101
rtems_shell_rtems_main_pwd, 57
rtems_shell_rtems_main_queue, 84
rtems_shell_rtems_main_region, 86
rtems_shell_rtems_main_rmdir, 58
rtems_shell_rtems_main_route, 98
rtems_shell_rtems_main_sema, 85
rtems_shell_rtems_main_setenv, 23
rtems_shell_rtems_main_sleep, 19
rtems_shell_rtems_main_stackuse, 76
rtems_shell_rtems_main_task, 83
rtems_shell_rtems_main_time, 24
rtems_shell_rtems_main_tty, 20
rtems_shell_rtems_main_umask, 60
rtems_shell_rtems_main_unmount, 61
rtems_shell_rtems_main_unsetenv, 23
rtems_shell_rtems_main_wdump, 65
rtems_shell_rtems_main_whoami, 21
rtems_shell_rtems_main_wkspc, 80

sema, 85
setenv, 22
shutdown, 74
sleep, 18
stackuse, 75

task, 83
time, 24
tty, 20

umask, 60
unmount, 60
unsetenv, 23

wdump, 64
whoami, 21
wkspc, 79