NAME

rotctld − Hamlib TCP rotator control daemon

SYNOPSIS

rotctld [*OPTION*]...

DESCRIPTION

The **rotctld** program is an NEW **Hamlib** rotator control daemon ready for testing that handles client requests via TCP sockets. This allows multiple user programs to share one rotator (this needs testing). Multiple rotators can be controlled on different TCP ports by use of multiple **rotctld** processes. The syntax of the commands are the same as **rotctl**. It is hoped that **rotctld** will be especially useful for client authors using languages such as Perl, Python, PHP, and others.

rotctld communicates to a client through a TCP socket using text commands shared with **rotctl**. The protocol is simple, commands are sent to **rotctld** on one line and **rotctld** responds to "get" commands with the requested values, one per line, when successful, otherwise, it responds with one line "RPRT x", where x is a negative number indicating the error code. Commands that do not return values respond with the line "RPRT x", where x is zero when successful, otherwise is a regative number indicating the error code. Each line is terminated with a newline '\n' character. This protocol is primarily for use by the *NET rotctl* (rot model 2) backend.

A separate **Extended Response** protocol extends the above behavior by echoing the received command string as a header, any returned values as a key: value pair, and the "RPRT x" string as the end of response marker which includes the **Hamlib** success or failure value. See the *PROTOCOL* section for details. Consider using this protocol for clients that will interact with **rotctld** directly through a TCP socket.

Keep in mind that **Hamlib** is BETA level software. While a lot of backend libraries lack complete rotator support, the basic functions are usually well supported. The API may change without publicized notice, while an advancement of the minor version (e.g. 1.1.x to 1.2.x) indicates such a change.

Please report bugs and provide feedback at the e-mail address given in the REPORTING BUGS section. Patches and code enhancements are also welcome.

OPTIONS

This program follows the usual GNU command line syntax, with long options starting with two dashes ('-').

Here is a summary of the supported options:

−m, --model=id
Select rotator model number. See -l, "list" option below.

−r, --rot-file=device
Use *device* as the file name of the port the rotator is connected. Often a serial port, but could be a USB to serial adapter or USB port device. Typically /dev/ttyS0, /dev/ttyS1, /dev/ttyUSB0, etc. on Linux or COM1, COM2, etc. on Win32.

−s, --serial-speed=baud
Set serial speed to *baud* rate. Uses maximum serial speed from rotor backend capabilities (set by -m above) as the default.

−T, --listen-addr=IPADDR
Use *IPADDR* as the listening IP address. The default is ANY.

−t, --port=number
Use *number* as the TCP listening port. The default is 4533.

N.B.: As **rigctld**'s default port is 4532, it is advisable to use odd numbered ports for **rotctld**, e.g. 4533, 4535, 4537, etc.

−L, --show-conf
List all config parameters for the rotator defined with -m above.

−C, --set-conf=parm=val[,parm=val]*
Set config parameter. e.g. --set-conf=stop_bits=2

Use -L option for a list.

−l, --list
List all model numbers defined in **Hamlib** and exit. As of 1.2.15.1 the list is sorted by model number.

N.B. In Linux the list can be scrolled back using Shift-PageUp/ Shift-PageDown, or using the scrollbars of a virtual terminal in X or the cmd window in Windows. The output can be piped to 'more' or 'less', e.g. 'rotctld -l | more'.

−u, --dump-caps
Dump capabilities for the radio defined with -m above and exit.

−e, --end-marker
Use END marker in rotctld protocol.

N.B.: This option should be considered obsolete. Please consider using the Extended Response protocol instead (see *PROTOCOL* below). This option will be removed in a future Hamlib release.

−v, --verbose
Set verbose mode, cumulative (see DIAGNOSTICS below).

−h, --help
Show a summary of these options and exit.

−V, --version
Show the version of **rotctld** and exit.

N.B. Some options may not be implemented by a given backend and will return an error. This is most likely to occur with the *−−set-conf* and *−−show-conf* options.

Please note that the backend for the rotator to be controlled, or the rotator itself may not support some commands. In that case, the operation will fail with a **Hamlib** error code.

COMMANDS

Commands can be sent over the TCP socket either as a single char, or as a long command name plus the value(s) space separated on one '\n' terminated line. See *PROTOCOL*.

Since most of the **Hamlib** operations have a *set* and a *get* method, an upper case letter will be used for *set* methods whereas the corresponding lower case letter refers to the *get* method. Each operation also has a long name; prepend a backslash to send a long command name.

Example (Perl): 'print $socket "\\dump_caps\n";' to see what the rotor's backend can do (NOTE: In Perl and many other languages a '\' will need to be escaped with a preceding '\' so that even though two backslash characters appear in the code, only one will be passed to **rotctld**. This is a possible bug, beware!).

Please note that the backend for the rotator to be controlled, or the rotator itself may not support some commands. In that case, the operation will fail with a **Hamlib** error message.

Here is a summary of the supported commands (In the case of "set" commands the quoted string is replaced by the value in the description. In the case of "get" commands the quoted string is the key name of the value returned.):

P, set_pos 'Azimuth' 'Elevation'
Set position: Azimuth and Elevation as double precision floating point values.

p, get_pos
Get position: 'Azimuth' and 'Elevation' as double precision floating point values.

M, move 'Direction' 'Speed'
Move the rotator in a specific direction at the given rate.

Values are integers where Direction is defined as 2 = Up, 4 = Down, 8 = Left, and 16 = Right. Speed is an integer between 1 and 100. Not all backends that implement the move command use the Speed value. At this time only the gs232a utilizes the Speed parameter.

S, stop Stop the rotator.

K, park
Park the antenna.

C, set_conf 'Token' 'Value'
Set Token to Value.

Backend dependent. Needs testing.

R, reset 'Reset'
Reset the rotator.

Integer value of '1' for Reset All.

_, get_info
Get misc information about the rotator.

At the moment returns 'Model Name'.

w, send_cmd 'Cmd'
Send raw command string to rotator.

For binary protocols enter values as \0xAA\0xBB. Expect a 'Reply' from the rotator which will likely be a binary block or an ASCII string.

Locator Commands

These commands offer conversions of Degrees Minutes Seconds to other formats, Maidenhead square locator conversions and distance and azimuth conversions.

L, lonlat2loc 'Longitude' 'Latitude' 'Loc Len [2-12]'
Returns the Maidenhead locator for the given 'Longitude' and 'Latitude'.

Both are floating point values. The precision of the returned square is controlled by 'Loc Len' which should be an even numbered integer value between 2 and 12.

For example, "+L -170.000000 -85.000000 12\n" returns "Locator: AA55AA00AA00\n".

l, loc2lonlat 'Locator'
Returns 'Longitude' and 'Latitude' in decimal degrees at the approximate center of the requested grid square (despite the use of double precision variables internally, some rounding error occurs). West longitude is expressed as a negative value. South latitude is expressed as a negative value. Locator can be from 2 to 12 characters in length.

For example, "+l AA55AA00AA00\n" returns "Longitude: -169.999983\nLatitude: -84.999991\n".

D, dms2dec 'Degrees' 'Minutes' 'Seconds' 'S/W'
Returns 'Dec Degrees', a signed floating point value.

Degrees and Minutes are integer values and Seconds is a floating point value. S/W is a flag with '1' indicating South latitude or West longitude and '0' North or East (the flag is needed as computers don't recognize a signed zero even though only the Degrees value only is typically signed in DMS notation).

d, dec2dms 'Dec Degrees'
Returns 'Degrees' 'Minutes' 'Seconds' 'S/W'.

Values are as in dms2dec above.

E, dmmm2dec 'Degrees' 'Dec Minutes' 'S/W'
Returns 'Dec Degrees', a signed floating point value.

Degrees is an integer value and Minutes is a floating point value. S/W is a flag with '1' indicating South latitude or West longitude and '0' North or East (the flag is needed as computers don't recognize a signed zero even though only the Degrees value only is typically signed in DMS notation).

e, dec2dmmm 'Dec Deg'
Returns 'Degrees' 'Minutes' 'S/W'.

Values are as in dmmm2dec above.

B, qrb 'Lon 1' 'Lat 1' 'Lon 2' 'Lat 2'
Returns 'Distance' 'Azimuth' where Distance is in km and Azimuth is in degrees.

All Lon/Lat values are signed floating point numbers.

A, a_sp2a_lp 'Short Path Deg'
Returns 'Long Path Deg' or -RIG_EINVAL upon input error..

Both are floating point values within the range 0.00 to 360.00.

a, d_sp2d_lp 'Short Path km'
Returns 'Long Path km'.

Both are floating point values.

PROTOCOL

Default Protocol

The **rotctld** protocol is intentionally simple. Commands are entered on a single line with any needed values. In Perl, reliable results are obtained by terminating each command string with a newline character, '\n'.

Example *set* (Perl code):

```
print $socket "P 135 10\n";

print $socket "\\set_pos 135 10\n";   # escape leading '\'
```

A one line response will be sent as a reply to *set* commands, "RPRT *x*\n" where *x* is the Hamlib error code with '0' indicating success of the command.

Responses from **rotctld** *get* commands are text values and match the same tokens used in the *set* commands. Each value is returned on its own line. On error the string "RPRT *x*\n" is returned where *x* is the Hamlib error code.

Example *get* (Perl code):

```
print $socket "p\n";
"135"
"10"
```

Most *get* functions return one to three values. A notable exception is the *dump_caps* function which returns many lines of key:value pairs.

This protocol is primarily used by the *NET rotctl* (rotctl model 2) backend which allows applications already written for Hamlib's C API to take advantage of **rotctld** without the need of rewriting application code. An application's user can select rotor model 2 ("NET rotctl") and then set rot_pathname to "localhost:4533" or other network host:port.

Extended Response Protocol

An *EXPERIMENTAL* Extended Response protocol has been introduced into **rotctld** as of February 10, 2010. This protocol adds several rules to the strings returned by **rotctld** and adds a rule for the command syntax.

1. The command received by **rotctld** is echoed with its long command name followed by the value(s) (if any) received from the client terminated by the specified response separator as the first record of the response.

2. The last record of each block is the string "RPRT *x*\n" where *x* is the numeric return value of the Hamlib backend function that was called by the command.

3. Any records consisting of data values returned by the rotor backend are prepended by a string immediately followed by a colon then a space and then the value terminated by the response separator. e.g. "Azimuth: 90.000000\n" when the command was prepended by '+'.

4. All commands received will be acknowledged by **rotctld** with records from rules 1 and 2. Records from rule 3 are only returned when data values must be returned to the client.

An example response to a *+P* command (note the prepended '+'):

```
$ echo "+P 90 45" | nc -w 1 localhost 4533
set_pos: 90 45
RPRT 0
```

In this case the long command name and values are returned on the first line and the second line contains the end of block marker and the numeric rig backend return value indicating success.

An example response to a *+\get_pos* query:

```
$ echo "+\get_pos" | nc -w 1 localhost 4533
get_pos:
Azimuth: 90.000000
Elevation: 45.000000
RPRT 0
```

In this case, as no value is passed to **rotctld**, the first line consists only of the long command name. The final line shows that the command was processed successfully by the rotor backend.

Invoking the Extended Response protocol requires prepending a command with a punctuation character. As shown in the examples above, prepending a '+' character to the command results in the responses being separated by a newline character ('\n'). Any other punctuation character recognized by the C *ispunct()* function except '\', '?', or '_' will cause that character to become the response separator and the entire response will be on one line.

Separator character summary:

'+'

Each record of the response is appended with a newline ('\n').

';', '|', or ','

Each record of the response is appended by the given character resulting in entire response on one line.

Common record separators for text representations of spreadsheet data, etc.

'?'

Reserved for 'help' in rotctl short command

'_'

Reserved for \get_info short command

'#'

Reserved for comments when reading a command file script

Other punctuation characters have not been tested! Use at your own risk.

For example, invoking a *;\get_pos* query with a leading ';' returns:

get_pos:;Azimuth: 90.000000;Elevation: 45.000000;RPRT 0

Or, using the pipe character '|' returns:

get_pos:|Azimuth: 90.000000|Elevation: 45.000000|RPRT 0

And a \set_pos command prepended with a '|' returns:

set_pos: 135 22.5|RPRT 0

Such a format will allow reading a response as a single event using a prefered response separator. Other punctuation characters have not been tested!

All commands with the exception of *\set_conf* have been tested with the Extended Response protocol and the included **testrotctld.pl** script.

EXAMPLES

Start **rotctld** for a Ham IV rotor with the RotorEZ installed using a USB-to-serial adapter and backgrounding:

$ rotctld -m 401 -r /dev/ttyUSB1 &

Start **rotctld** for RotorEZ using COM2 on Win32:

$ rotctl -m 401 -r COM2

Connect to the already running **rotctld**, and set position to 135.0 degrees azimuth and 30.0 degrees elevation with a 1 second read timeout:

$ echo "\set_pos 135.0 30.0" | nc -w 1 localhost 4533

Connect to a running **rotctld** with **rotctl** on the local host:

$ rotctl -m2

DIAGNOSTICS

The **-v**, **--version** option allows different levels of diagnostics to be output to **stderr** and correspond to -v for BUG, -vv for ERR, -vvv for WARN, -vvvv for VERBOSE, or -vvvvv for TRACE.

A given verbose level is useful for providing needed debugging information to the email address below. For example, TRACE output shows all of the values sent to and received from the rotator which is very useful for rotator backend library development and may be requested by the developers. See the **README.betatester** and **README.developer** files for more information.

SECURITY

No authentication whatsoever; DO NOT leave this TCP port open wide to the Internet. Please ask if stronger security is needed or consider using an SSH tunnel.

As **rotctld** does not need any greater permissions than **rotctl**, it is advisable to not start **rotctld** as *root* or another system user account in order to limit any vulnerability.

BUGS

The daemon is not detaching and backgrounding itself.

Much testing needs to be done.

REPORTING BUGS

Report bugs to <hamlib-developer@lists.sourceforge.net>.

We are already aware of the bugs in the previous section :-)

AUTHORS

Written by Stephane Fillod, Nate Bargmann, and the Hamlib Group

<http://www.hamlib.org>.

COPYRIGHT

SEE ALSO

rotctl(1), **hamlib**(3)