## NAME

rigctld − Hamlib TCP rig control daemon

## SYNOPSIS

**rigctld** [*OPTION*]...

## DESCRIPTION

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

**rigctld** communicates to a client through a TCP socket using text commands shared with **rigctl**. The protocol is simple; commands are sent to **rigctld** on one line and **rigctld** 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 rigctl* (rig 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 **rigctld** directly through a TCP socket.

Keep in mind that **Hamlib** is BETA level software. While a lot of backend libraries lack complete rig 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 radio model number. See the -l, --list option below.

**−r, --rig-file=device**
Use *device* as the file name of the port the radio 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.

**−p, --ptt-file=device**
Use *device* as the file name of the Push-To-Talk device using a device file as described above.

**−d, --dcd-file=device**
Use *device* as the file name of the Data Carrier Detect device using a device file as described above.

**−P, --ptt-type=type**
Use *type* of Push-To-Talk device. Supported types are RIG (CAT command), DTR, RTS, PARALLEL, NONE.

**−D, --dcd-type=type**
Use *type* of Data Carrier Detect device. Supported types are RIG (CAT command), DSR, CTS, CD, PARALLEL, NONE.

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

**−c, --civaddr=id**
Use *id* as the CI-V address to communicate with the rig. Only useful for Icom rigs.

**N.B.**: The *id* is in decimal notation, unless prefixed by *0x* for a hexadecimal value.

**−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 4532.

**N.B.**: As **rotctld**'s default port is 4533, it is advisable to use even numbered ports for **rigctld**, e.g. 4532, 4534, 4536, etc.

**−L, --show-conf**
List all config parameters for the radio 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.

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

**−o, --vfo**
Set vfo mode, requiring an extra VFO argument in front of each appropriate command (except *\set_vfo*!). Otherwise, 'currVFO' is assumed when this option is not set and an extra VFO argument is not used. See *\chk_vfo* below.

**−e, --end-marker**
Use END marker in rigctld 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 **rigctld** 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 radio to be controlled, or the radio 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 radio’s backend can do
(**N.B.**: 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 **rigctld**. This is a possible bug, beware!).

Please note that the backend for the radio to be controlled, or the radio 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.):

**F, set_freq ’Frequency’**
Set ’Frequency’, in Hz.

**f, get_freq**
Get ’Frequency’, in Hz.

**M, set_mode ’Mode’ ’Passband’**
Set ’Mode’: USB, LSB, CW, CWR, RTTY, RTTYR, AM, FM, WFM, AMS, PKTLSB, PKTUSB, PKTFM, ECSSUSB, ECSSLSB, FAX, SAM, SAL, SAH, DSB.

Set ’Passband’ in Hz, or ’0’ for the Hamlib backend default.

**m, get_mode**
Get ’Mode’ ’Passband’.

Returns Mode as a string from *set_mode* above and Passband in Hz.

**V, set_vfo ’VFO’**
Set ’VFO’: VFOA, VFOB, VFOC, currVFO, VFO, MEM, Main, Sub, TX, RX.

In VFO mode only a single VFO parameter is required.

**v, get_vfo**
Get current ’VFO’.

Returns VFO as a string from *set_vfo* above.

**J, set_rit ’RIT’**
Set ’RIT’, in Hz, can be + or -.

A value of ’0’ resets RIT and *should* turn RIT off. If not, file a bug report against the Hamlib backend.

**j, get_rit**
Get ’RIT’, in Hz.

**Z, set_xit ’XIT’**
Set ’XIT’, in Hz can be + or -.

A value of ’0’ resets RIT and *should* turn RIT off. If not, file a bug report against the Hamlib backend.

**z, get_xit**
Get ’XIT’, in Hz.

**T, set_ptt ’PTT’**
Set ’PTT’, 0 (RX) or 1 (TX).

**t, get_ptt**
Get ’PTT’ status.

**0x8b, get_dcd**
Get ’DCD’ (squelch) status, 0 (Closed) or 1 (Open)

**R, set_rptr_shift ’Rptr Shift’**
Set ’Rptr Shift’: "+", "-" or something else for none.

**r, get_rptr_shift**
Get ’Rptr Shift’. Returns "+", "-" or "None".

**O, set_rptr_offs ’Rptr Offset’**
Set ’Rptr Offset’, in Hz.

**o, get_rptr_offs**
Get ’Rptr Offset’, in Hz.

**C, set_ctcss_tone ’CTCSS Tone’**
Set ’CTCSS Tone’, in tenths of Hz.

**c, get_ctcss_tone**
Get ’CTCSS Tone’, in tenths of Hz.

**D, set_dcs_code ’DCS Code’**
Set ’DCS Code’.

**d, get_dcs_code**
Get ’DCS Code’.

**0x90, set_ctcss_sql ’CTCSS Sql’**
Set ’CTCSS Sql’ tone, in tenths of Hz.

**0x91, get_ctcss_sql**
Get ’CTCSS Sql’ tone, in tenths of Hz.

**0x92, set_dcs_sql ’DCS Sql’**
Set ’DCS Sql’ code.

**0x93, get_dcs_sql**
Get ’DCS Sql’ code.

**I, set_split_freq ’Tx Frequency’**
Set ’TX Frequency’, in Hz.

**i, get_split_freq**
Get ’TX Frequency’, in Hz.

**X, set_split_mode ’TX Mode’ ’TX Passband’**
Set ’TX Mode’: AM, FM, CW, CWR, USB, LSB, RTTY, RTTYR, WFM, AMS, PKTLSB, PKTUSB, PKTFM, ECSSUSB, ECSSLSB, FAX, SAM, SAL, SAH, DSB.

The ’TX Passband’ is the exact passband in Hz, or ’0’ for the Hamlib backend default.

**x, get_split_mode**
Get ’TX Mode’ and ’TX Passband’.

Returns TX mode as a string from *set_split_mode* above and TX passband in Hz.

**S, set_split_vfo ’Split’ ’TX VFO’**
Set ’Split’ mode, ’0’ or ’1’, and ’TX VFO’ from *set_vfo* above.

**s, get_split_vfo**
Get ’Split’ mode, ’0’ or ’1’, and ’TX VFO’.

**N, set_ts 'Tuning Step'**
Set 'Tuning Step', in Hz.

**n, get_ts**
Get 'Tuning Step', in Hz.

**U, set_func 'Func' 'Func Status'**
Set 'Func' 'Func Status'.

Func is one of: FAGC, NB, COMP, VOX, TONE, TSQL, SBKIN, FBKIN, ANF, NR, AIP, APF, MON, MN, RF, ARO, LOCK, MUTE, VSC, REV, SQL, ABM, BC, MBC, AFC, SATMODE, SCOPE, RESUME, TBURST, TUNER.

Func Status argument is a non null value for "activate", "de-activate" otherwise, much as TRUE/FALSE definitions in C language.

**u, get_func**
Get 'Func' 'Func Status'.

Returns Func as a string from *set_func* above and Func status as a non null value.

**L, set_level 'Level' 'Level Value'**
Set 'Level' and 'Level Value'.

Level is one of: PREAMP, ATT, VOX, AF, RF, SQL, IF, APF, NR, PBT_IN, PBT_OUT, CWPITCH, RFPOWER, MICGAIN, KEYSPD, NOTCHF, COMP, AGC, BKINDL, BAL, METER, VOXGAIN, ANTIVOX, SLOPE_LOW, SLOPE_HIGH, RAWSTR, SQLSTAT, SWR, ALC, STRENGTH.

The Level Value can be a float or an integer.

**l, get_level**
Get 'Level' 'Level Value'.

Returns Level as a string from *set_level* above and Level value as a float or integer.

**P, set_parm 'Parm' 'Parm Value'**
Set 'Parm' 'Parm Value'

Parm is one of: ANN, APO, BACKLIGHT, BEEP, TIME, BAT, KEYLIGHT.

**p, get_parm**
Get 'Parm' 'Parm Value'.

Returns Parm as a string from *set_parm* above and Parm Value as a float or integer.

**B, set_bank 'Bank'**
Set 'Bank'. Sets the current memory bank number.

**E, set_mem 'Memory#'**
Set 'Memory#' channel number.

**e, get_mem**
Get 'Memory#' channel number.

**G, vfo_op 'Mem/VFO Op'**
Perform 'Mem/VFO Op'.

Mem VFO operation is one of: CPY, XCHG, FROM_VFO, TO_VFO, MCL, UP, DOWN, BAND_UP, BAND_DOWN, LEFT, RIGHT, TUNE, TOGGLE.

**g, scan 'Scan Fct' 'Scan Channel'**
Perform 'Scan Fct' 'Scan Channel'.

Scan function/channel is one of: STOP, MEM, SLCT, PRIO, PROG, DELTA, VFO, PLT.

**H, set_channel 'Channel'**
Set memory 'Channel' data. Not implemented yet.

**h, get_channel**
Get memory 'Channel' data. Not implemented yet.

**A, set_trn 'Transceive'**
Set 'Transceive' mode (reporting event): OFF, RIG, POLL.

**a, get_trn**
Get 'Transceive' mode (reporting event) as in *set_trn* above.

**Y, set_ant 'Antenna'**
Set 'Antenna' number (0, 1, 2, ..).

**y, get_ant**
Get 'Antenna' number (0, 1, 2, ..).

***, reset 'Reset'**
Perform rig 'Reset'.

0 = None, 1 = Software reset, 2 = VFO reset, 4 = Memory Clear reset, 8 = Master reset. Since these values are defined as a bitmask in rig.h, it should be possible to AND these values together to do multiple resets at once, if the backend supports it or supports a reset action via rig control at all.

**b, send_morse 'Morse'**
Send 'Morse' symbols.

**0x87, set_powerstat 'Power Status'**
Set power On/Off/Standby 'Power Status'.

0 = Power Off, 1 = Power On, 2 = Power Standby. Defined as a bitmask in rig.h.

**0x88, get_powerstat**
Get power On/Off/Standby 'Power Status' as in *set_powerstat* above.

**0x89, send_dtmf 'Digits'**
Set DTMF 'Digits'.

**0x8a, recv_dtmf**
Get DTMF 'Digits'.

**_, get_info**
Get misc information about the rig (no VFO in 'VFO mode' or value is passed).

**1, dump_caps**
Not a real rig remote command, it just dumps capabilities, i.e. what the backend knows about this model, and what it can do. TODO: Ensure this is in a consistent format so it can be read into a hash, dictionary, etc. Bug reports requested.

**N.B.**: This command will produce many lines of output so be very careful if using a fixed length array! For example, running this command against the Dummy backend results in over 5kB of text output.

VFO parameter not used in 'VFO mode'.

**2, power2mW ’Power [0.0..1.0]’ ’Frequency’ ’Mode’**
Returns ’Power mW’

Converts a Power value in a range of *0.0 ... 1.0* to the real transmit power in milli-Watts (integer). The *frequency* and *mode* also need to be provided as output power may vary according to these values.

VFO parameter not used in ’VFO mode’.

**4, mW2power ’Power mW’ ’Frequency’ ’Mode’**
Returns ’Power [0.0..1.0]’

Converts the real transmit power in milli-Watts (integer) to a Power value in a range of *0.0 ... 1.0*. The *frequency* and *mode* also need to be provided as output power may vary according to these values.

VFO parameter not used in ’VFO mode’.

**w, send_cmd ’Cmd’**
Send raw command string to rig.

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

**chk_vfo**
Returns "CHKVFO 1\n" (single line only) if **rigctld** was invoked with the *-o* or *--vfo* option, "CHKVFO 0\n" if not.

When in VFO mode the client will need to pass ’VFO’ as the first parameter to *\set* or *\get* commands. ’VFO’ is one of the strings defined for *\set_vfo* above.

## PROTOCOL

### Default Protocol

The **rigctld** 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 "F 14250000\n";
print $socket "\\set_mode LSB 2400\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 **rigctld** *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 "f\n";
"14250000\n"
```

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 rigctl* (rigctl model 2) backend which allows applications already written for Hamlib’s C API to take advantage of **rigctld** without the need of rewriting application code. An application’s user can select rig model 2 ("NET rigctl") and then set rig_pathname to

"localhost:4532" or other network host:port.

**Extended Response Protocol**

An *EXPERIMENTAL* Extended Response protocol has been introduced into **rigctld** as of February 16, 2010. This protocol adds several rules to the strings returned by **rigctld** and adds a rule for the command syntax.

1. The command received by **rigctld** 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 record line of the response.

2. The last line 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 rig 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. "Frequency: 14250000\n" when the command was prepended by '+'.

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

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

```
$ echo "+M USB 2400" | nc -w 1 localhost 4532
set_mode: USB 2400
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_mode* query:

```
$ echo "+\get_mode" | nc -w 1 localhost 4532
get_mode:
Mode: USB
Passband: 2400
RPRT 0
```

In this case, as no value is passed to **rigctld**, the first line consists only of the long command name. The final line shows that the command was processed successfully by the rig 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 rigctl 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_mode* query with a leading ';' returns:

get_mode:;Mode: USB;Passband: 2400;RPRT 0

Or, using the pipe character '|' returns:

get_mode:|Mode: USB|Passband: 2400|RPRT 0

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

set_mode: USB 2400|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!

The following commands have been tested with the Extended Response protocol and the included **testctld.pl** script:
*\set_freq \get_freq \set_split_freq \get_split_freq*
*\set_mode \get_mode \set_split_mode \get_split_mode*
*\set_vfo \get_vfo \set_split_vfo \get_split_vfo*
*\set_rit \get_rit*
*\set_xit \get_xit*
*\set_ptt \get_ptt*
*\power2mW \mW2power*
*\dump_caps*

## EXAMPLES

Start **rigctld** for a Yaesu FT-920 using a USB-to-serial adapter and backgrounding:

$ rigctld -m 114 -r /dev/ttyUSB1 &

Start **rigctld** for a Yaesu FT-920 using a USB to serial adapter while setting baud rate and stop bits, and backgrounding:

$ rigctld -m 114 -r /dev/ttyUSB1 -s 4800 -C stop_bits=2 &

Start **rigctld** for an Elecraft K3 using COM2 on Win32:

$ rigctld -m 229 -r COM2

Connect to the already running **rigctld**, and set current frequency to 14.266 MHz with a 1 second read timeout using the default protocol:

$ echo "\set_freq 14266000" | nc -w 1 localhost 4532

## DIAGNOSTICS

The **-v**, **--verbose**, 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 radio which is very useful for radio 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 **rigctld** does not need any greater permissions than **rigctl**, it is advisable to not start **rigctld** 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

**rigctl**(1), **hamlib**(3)