RIGCTL

NAME

rigctl - control radio transceivers and receivers

SYNOPSIS

rigctl [**-hiIlLnouV**] [**-m** *id*] [**-r** *device*] [**-p** *device*] [**-d** *device*] [**-P** *type*] [**-D** *type*] [**-s** *baud*] [**-c** *id*] [**-t** *char*] [**-C** *parm=val*] [**-v**[**-Z**]] [**command**|-]

DESCRIPTION

Control radio transceivers and receivers. **rigctl** accepts **commands** from the command line as well as in interactive mode if none are provided on the command line.

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.

Please report bugs and provide feedback at the e-mail address given in the **BUGS** section below. Patches and code enhancements sent to the same address are welcome.

OPTIONS

This program follows the usual GNU command line syntax. Short options that take an argument may have the value follow immediately or be separated by a space. Long options starting with two dashes (’-’) require an ’=’ between the option and any argument.

Here is a summary of the supported options:

-m, **--model**=*id*

Select radio model number.

See model list (use “rigctl -l”).

Note: **rigctl** (or third party software using the C API) will use radio model 2 for **NET rigctl** (communicating with **rigctld**).

-r, **--rig-file**=*device*

Use *device* as the file name of the port connected to the radio.

Often a serial port, but could be a USB to serial adapter. Typically */dev/ttyS0*, */dev/ttyS1*, */dev/ttyUSB0*, etc. on Linux, *COM1*, *COM2*, etc. on MS Windows. The BSD flavors and Mac OS/X have their own designations. See your system's documentation.

Can be a network address:port, e.g. 127.0.0.1:12345

The special string "uh-rig" may be given to enable micro-ham device support.

-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', 'CM108', 'GPIO', 'GPION', 'NONE', overriding PTT type defined in the rig's backend.

Some side effects of this command are that when type is set to DTR, read PTT state comes from the **Hamlib** frontend, not read from the radio. When set to NONE, PTT state cannot be read or set even if rig backend supports reading/setting PTT status from the rig.

-D, **--dcd-type**=*type*

Use *type* of Data Carrier Detect device.

Supported types are 'RIG' (CAT command), 'DSR', 'CTS', 'CD', 'PARALLEL', 'CM108', 'GPIO', 'GPION', 'NONE'.

-s, **--serial-speed**=*baud*

Set serial speed to *baud* rate.

Uses maximum serial speed from radio 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 and some Ten-Tec rigs.

Note: The *id* is in decimal notation, unless prefixed by *0x*, in which case it is hexadecimal.

-t, **--send-cmd-term**=*char*

Change the termination *char* for text protocol when using the **send_cmd** command.

The default value is ASCII CR ('0x0D'). ASCII non-printing characters can be given as the ASCII number in hexadecimal format prepended with "0x". You may pass an empty string for no termination char. The string "-1" tells **rigctl** to switch to binary protocol. See the **send_cmd** command for further explanation.

For example, to specify a command terminator for Kenwood style text commands pass "-t ';'" to rigctl. See **EXAMPLE** below.

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

-C, **--set-conf**=*parm=val*[,*parm=val*]
Set radio configuration parameter(s), e.g. *stop_bits=2*.

Use the **-L** option above for a list of configuration parameters for a given model number.

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

-l, **--list**
List all model numbers defined in **Hamlib** and exit.

The list is sorted by model number.

Note: 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**(1) or **less**(1), e.g. “rigctl -l | more”.

-o, **--vfo**
Enable vfo mode.

An extra VFO argument will be required in front of each appropriate command (except **set_vfo**). Otherwise, ’currVFO’ is used when this option is not set and an extra VFO argument is not used.

-n, **--no-restore-ai**
On exit **rigctl** restores the state of auto information (AI) on the controlled rig.

If this is not desired, for example if you are using **rigctl** to turn AI mode on or off, pass this option.

-i, **--read-history**
Read previously saved command and argument history from a file (default *$HOME/.rigctl_history*) for the current session.

Available when **rigctl** is built with Readline support (see READLINE below).

Note: To read a history file stored in another directory, set the **RIGCTL_HIST_DIR** environment variable, e.g. “RIGCTL_HIST_DIR=~/tmp rigctl -i”. When RIGCTL_HIST_DIR is not set, the value of **HOME** is used.

-I, **--save-history**
Write current session (and previous session(s), if **-i** option is given) command and argument history to a file (default *$HOME/.rigctl_history*) at the end of the current session.

Complete commands with arguments are saved as a single line to be recalled and used or edited. Available when **rigctl** is built with Readline support (see **READLINE** below).

Note: To write a history file in another directory, set the **RIGCTL_HIST_DIR** environment variable, e.g. “RIGCTL_HIST_DIR=~/tmp rigctl -IRq. When RIGCTL_HIST_DIR is not set, the value of **HOME** is used.

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

-Y,**--ignore-err**
Ignores rig open errors

-Z, **--debug-time-stamps**
Enable time stamps for the debug messages.

Use only in combination with the **-v** option as it generates no output on its own.

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

-V, **--version**
Show version of **rigctl** and exit.

- Stop option processing and read commands from standard input.

See **Standard Input** below.

Note: 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 entered either as a single char, or as a long command name. The commands are not prefixed with a dash as the options are. They may be typed in when in interactive mode or provided as argument(s) in command line interface mode. In interactive mode commands and their arguments may be entered on a single line:

M LSB 2400

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

Example: Use "\dump_caps" to see what capabilities this radio and backend support.

Note: 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.

Standard Input
As an alternative to the **READLINE** interactive command entry or a single command for each run, **rigctl** features a special option where a single dash ('-') may be used to read commands from standard input (**stdin**). Commands must be separated by whitespace similar to the commands given on the command line. Comments may be added using the '#' character, all text up until the end of the current line including the '#' character is ignored.

A simple example (typed text is in bold):

$ **cat <<.EOF. >cmds.txt**
> **# File of commands**
> **v f m**
 # query
rig
> **V** **# set rig**
VFOB F
14200000
M CW
500
> **v f m**

```
        # query
        rig
> .EOF.

$ rigctl -m1 - <cmds.txt

v VFOA

f 145000000

m FM
15000

V VFOB
F 14200000
M CW 500
v VFOB

f 14200000

m CW
500

$
```

rigctl Commands

A summary of commands is included below (In the case of **set** commands the quoted italicized string is replaced by the value in the description. In the case of **get** commands the quoted italicized string is the key name of the value returned.):

Q|**q**, exit rigctl

Exit rigctl in interactive mode.

When rigctl is controlling the rig directly, will close the rig backend and port. When rigctl is connected to rigctld (radio model 2), the TCP/IP connection to rigctld is closed and rigctld remains running, available for another TCP/IP network connection.

F, **set_freq** '*Frequency*'

Set '*Frequency*', in Hz.

Frequency may be a floating point or integer value.

f, **get_freq**

Get '*Frequency*', in Hz.

Returns an integer value and the VFO hamlib thinks is active. Note that some rigs (e.g. all Icoms) cannot track current VFO so hamlib can get out of sync with the rig if the user presses rig buttons like the VFO.

M, **set_mode** '*Mode*' '*Passband*'

Set '*Mode*' and '*Passband*'.

Mode is a token: ’USB’, ’LSB’, ’CW’, ’CWR’, ’RTTY’, ’RTTYR’, ’AM’, ’FM’, ’WFM’, ’AMS’, ’PKTLSB’, ’PKTUSB’, ’PKTFM’, ’ECSSUSB’, ’ECSSLSB’, ’FA’, ’SAM’, ’SAL’, ’SAH’, ’DSB’.

Passband is in Hz as an integer, -1 for no change, or ’0’ for the radio backend default.

Note: Passing a ’?’ (query) as the first argument instead of a Mode token will return a space separated list of radio backend supported Modes. Use this to determine the

supported Modes of a given radio backend.

m, **get_mode**

Get '*Mode*' and '*Passband*'.

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

V, **set_vfo** '*VFO*'

Set '*VFO*'.

VFO is a token: ’VFOA’, ’VFOB’, ’VFOC’, ’currVFO’, ’VFO’, ’MEM’, ’Main’, ’Sub’, ’TX’, ’RX’.

In VFO mode (see **--vfo** option above) only a single VFO parameter is required:

$ **rigctl -m 229 -r /dev/rig -o**

Rig command: V
VFO: VFOB

Rig command:

v, **get_vfo**

Get current '*VFO*'.

Returns VFO as a token as in **set_vfo** above.

J, **set_rit** '*RIT*'

Set '*RIT*'.

RIT is in Hz and can be + or -. A value of ’0’ resets RIT (Receiver Incremental Tuning) to match the VFO frequency.

Note: RIT needs to be explicitly activated or deactivated with the **set_func** command. This allows setting the RIT offset independently of its activation and allows RIT to remain active while setting the offset to ’0’.

j, **get_rit**

Get '*RIT*' in Hz.

Returned value is an integer.

Z, **set_xit** '*XIT*'

Set '*XIT*'.

XIT is in Hz and can be + or -. A value of ’0’ resets XIT (Transmitter Incremental Tuning) to match the VFO frequency.

Note: XIT needs to be explicitly activated or deactivated with the **set_func** command. This allows setting the XIT offset independently of its activation and allows XIT to remain active while setting the offset to ’0’.

z, **get_xit**

Get '*XIT*' in Hz.

Returned value is an integer.

T, **set_ptt** '*PTT*'

Set '*PTT*'.

PTT is a value: ’0’ (RX), ’1’ (TX), ’2’ (TX mic), or ’3’ (TX data).

t, **get_ptt**

Get '*PTT*' status.

Returns PTT as a value in **set_ptt** above.

S, **set_split_vfo** '*Split*' '*TX VFO*'

Set '*Split*' mode.

Split is either ’0’ = Normal or ’1’ = Split.

Set '*TX VFO*'.

TX VFO is a token: ’VFOA’, ’VFOB’, ’VFOC’, ’currVFO’, ’VFO’, ’MEM’, ’Main’, ’Sub’, ’TX’, ’RX’.

s, **get_split_vfo**

Get '*Split*' mode.

Split is either ’0’ = Normal or ’1’ = Split.

Get '*TX VFO*'.

TX VFO is a token as in **set_split_vfo** above.

I, **set_split_freq** '*Tx Frequency*'

Set '*TX Frequency*', in Hz.

Frequency may be a floating point or integer value.

i, **get_split_freq**

Get '*TX Frequency*', in Hz.

Returns an integer value.

X, **set_split_mode** '*TX Mode*' '*TX Passband*'

Set '*TX Mode*' and '*TX Passband*'.

TX Mode is a token: ’USB’, ’LSB’, ’CW’, ’CWR’, ’RTTY’, ’RTTYR’, ’AM’, ’FM’, ’WFM’, ’AMS’, ’PKTLSB’, ’PKTUSB’, ’PKTFM’, ’ECSSUSB’, ’ECSSLSB’, ’FA’, ’SAM’, ’SAL’, ’SAH’, ’DSB’.

TX Passband is in Hz as an integer, or ’0’ for the radio backend default.

Note: Passing a ’?’ (query) as the first argument instead of a TX Mode token will return a space separated list of radio backend supported TX Modes. Use this to determine the supported TX Modes of a given radio backend.

x, **get_split_mode**

Get '*TX Mode*' and '*TX Passband*'.

Returns TX Mode as a token and TX Passband in Hz as in **set_split_mode** above.

Y, **set_ant** '*Antenna*' '*Option*'

Set '*Antenna*' and '*Option*'.

Number is 1-based antenna# (’1’, ’2’, ’3’, ...).

Option depends on rig..for Icom it probably sets the Tx & Rx antennas as in the IC-7851. See your manual for rig specific option values. Most rigs don’t care about the option.

For the IC-7851, FTDX3000 (and perhaps others) it means this:

1 = TX/RX = ANT1 FTDX3000=ANT1/ANT3
2 = TX/RX = ANT2 FTDX3000=ANT2/ANT3
3 = TX/RX = ANT3 FTDX3000=ANT3

4 = TX/RX = ANT1/ANT4
5 = TX/RX = ANT2/ANT4
6 = TX/RX = ANT3/ANT4

y, **get_ant** '*Antenna*'
Get '*Antenna*'

A value of 0 for Antenna will return the current TX antenna

> 0 is 1-based antenna# (’1’, ’2’, ’3’, ...).

Option returned depends on rig..for Icom is likely the RX only flag.

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

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

R, **set_rptr_shift** '*Rptr Shift*'
Set '*Rptr Shift*'.

Rptr Shift is one of: ’+’, ’-’, 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.

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*' and '*Func Status*'.

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

Func Status is a non null value for “activate” or “de-activate” otherwise, much as TRUE/FALSE definitions in the C language (true is non-zero and false is zero, ’0’).

Note: Passing a ’?’ (query) as the first argument instead of a Func token will return a space separated list of radio backend supported set function tokens. Use this to determine the supported functions of a given radio backend.

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

Returns Func Status as a non null value for the Func token given as in **set_func** above.

Note: Passing a ’?’ (query) as the first argument instead of a Func token will return a space separated list of radio backend supported get function tokens. Use this to determine the supported functions of a given radio backend.

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

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

The Level Value can be a float or an integer value. For the AGC token the value is one of ’0’ = OFF, ’1’ = SUPERFAST, ’2’ = FAST, ’3’ = SLOW, ’4’ = USER, ’5’ = MEDIUM, ’6’ = AUTO.

Note: Passing a ’?’ (query) as the first argument instead of a Level token will return a space separated list of radio backend supported set level tokens. Use this to determine the supported levels of a given radio backend.

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

Returns Level Value as a float or integer for the Level token given as in **set_level** above.

Note: Passing a ’?’ (query) as the first argument instead of a Level token will return a space separated list of radio backend supported get level tokens. Use this to determine the supported levels of a given radio backend.

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

Parm is a token: ’ANN’, ’APO’, ’BACKLIGHT’, ’BEEP’, ’TIME’, ’BAT’, ’KEYLIGHT’.

Note: Passing a ’?’ (query) as the first argument instead of a Parm token will return a space separated list of radio backend supported set parameter tokens. Use this to determine the supported parameters of a given radio backend.

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

Returns Parm Value as a float or integer for the Parm token given as in **set_parm** above.

Note: Passing a '?' (query) as the first argument instead of a Parm token will return a space separated list of radio backend supported get parameter tokens. Use this to determine the supported parameters of a given radio backend.

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 a '*Mem/VFO Op*'.

Mem/VFO Operation is a token: 'CPY', 'XCHG', 'FROM_VFO', 'TO_VFO', 'MCL', 'UP', 'DOWN', 'BAND_UP', 'BAND_DOWN', 'LEFT', 'RIGHT', 'TUNE', 'TOGGLE'.

Note: Passing a '?' (query) as the first argument instead of a Mem/VFO Op token will return a space separated list of radio backend supported Set Mem/VFO Op tokens. Use this to determine the supported Mem/VFO Ops of a given radio backend.

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

Perform a '*Scan Fct*' on a '*Scan Channel*'.

Scan Function is a token: 'STOP', 'MEM', 'SLCT', 'PRIO', 'PROG', 'DELTA', 'VFO', 'PLT'.

Scan Channel is an integer (maybe?).

Note: Passing a '?' (query) as the first argument instead of a Scan Fct token will return a space separated list of radio backend supported Scan Function tokens. Use this to determine the supported Scan Functions of a given radio backend.

H, **set_channel** '*Channel*'

Set memory '*Channel*' data.

Sets memory channel information

h, **get_channel** '*readonly*'

Get channel memory.

If readonly!=0 then only channel data is returned and rig remains on the current channel. If readonly=0 then rig will be set to the channel requested. data.

A, **set_trn** '*Transceive*'

Set '*Transceive*' mode.

Transcieve is a token: 'OFF', 'RIG', 'POLL'.

Transceive is a mechanism for radios to report events without a specific call for information.

Note: Passing a '?' (query) as the first argument instead of a Transceive token will return a space separated list of radio backend supported Transceive mode tokens. Use this to determine the supported Transceive modes of a given radio backend.

a, **get_trn**

Get '*Transceive*' mode.

Transceive mode (reporting event) as in **set_trn** above.

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

Reset is a value: ’0’ = None, ’1’ = Software reset, ’2’ = VFO reset, ’4’ = Memory Clear reset, ’8’ = Master reset.

Since these values are defined as a bitmask in *include/hamlib/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.

0x87, **set_powerstat** '*Power Status*'
Set '*Power Status*'.

Power Status is a value: ’0’ = Power Off, ’1’ = Power On, ’2’ = Power Standby (enter standby), ’4’ = Power Operate (leave standby).

0x88, **get_powerstat**
Get '*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.

0xf5, **get_rig_info**
Get misc information about the rig vfo status and other info.

0xf3, **get_vfo_info** '*VFO*'
Get misc information about a specific vfo.

dump_state
Return certain state information about the radio backend.

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.

Note: 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).

'*Frequency*' and '*Mode*' also need to be provided as output power may vary according to these values.

VFO parameter is 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*.

'*Frequency*' and '*Mode*' also need to be provided as output power may vary according to these values.

VFO parameter is not used in VFO mode.

w, **send_cmd** '*Cmd*'
Send a raw command string to the radio.

This is useful for testing and troubleshooting radio commands and responses when developing a backend.

For binary protocols enter values as \0xAA\0xBB. Expect a '*Reply*' from the radio which will likely be a binary block or an ASCII string depending on the radio's protocol (see your radio's computer control documentation).

The command terminator, set by the **send-cmd-term** option above, will terminate each command string sent to the radio. This character should not be a part of the input string.

W, **send_cmd_rx** '*Cmd*' *nbytes*
Send a raw command string to the radio and expect nbytes returned.

This is useful for testing and troubleshooting radio commands and responses when developing a backend. If the # of bytes requested is <= the number actually returned no timeout will occur.

The command argument can have no spaces in it. For binary protocols enter values as \0xAA\0xBB. Expect a '*Reply*' from the radio which will likely be a binary block or an ASCII string depending on the radio's protocol (see your radio's computer control documentation).

The command terminator, set by the **send-cmd-term** option above, will terminate each command string sent to the radio. This character should not be a part of the input string.

chk_vfo
Get '*Status*'

Returns Status as 1 if vfo option is on and 0 if vfo option is off. This command reflects the -o switch for rigctl and ritctld and can be dynamically changed by **set_vfo_opt.**

set_vfo_opt '*Status*'
Set '*Status*'

Set vfo option Status 1=on or 0=off This is the same as using the -o switch for rigctl and ritctld. This can be dyamically changed while running.

pause '*Seconds*'
Pause for the given whole (integer) number of '*Seconds*' before sending the next command to the radio.

READLINE

If **Readline** library development files are found at configure time, **rigctl** will be conditonally built with Readline support for command and argument entry. Readline command key bindings are at their defaults as described in the Readline manual. **rigctl** sets the name "rigctl" which can be used in Conditional Init Constructs in the Readline Init File (*$HOME/.inputrc* by default) for custom keybindings unique to **rigctl**.

Command history is available with Readline support as described in the Readline History manual. Command and argument strings are stored as single lines even when arguments are prompted for input individually. Commands and arguments are not validated and are stored as typed with values separated by a single space.

Normally session history is not saved, however, use of either of the **-i**/**--read-history** or **-I**/**--save-history** options when starting **rigctl** will cause any previously saved history to be read in and/or the current and any previous session history (assuming the **-i** and **-I** options are given together) will be written out when **rigctl** is closed. Each option is mutually exclusive, i.e. either may be given separately or in combination. This is useful to save a set of commands and then read them later but not write the modified history for a consistent set of test commands in interactive mode, for example.

History is stored in *$HOME/.rigctl_history* by default although the destination directory may be changed by setting the **RIGCTL_HIST_DIR** environment variable. When RIGCTL_HIST_DIR is unset, the value of the **HOME** environment variable is used instead. Only the destination directory may be changed at this time.

If Readline support is not found at configure time the original internal command handler is used. Readline is not used for **rigctl** commands entered on the command line regardless if Readline support is built in or not.

Note: Readline support is not included in the MS Windows 32 or 64 bit binary builds supplied by the Hamlib Project. Running **rigctl** on the MS Windows platform in the 'cmd' shell does give session command line history, however, it is not saved to disk between sessions.

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.

EXIT STATUS

rigctl exits with:

0	if all operations completed normally;
1	if there was an invalid command line option or argument;
2	if an error was returned by **Hamlib**.

EXAMPLES

Start **rigctl** for a Yaesu FT-920 using a USB to serial adapter on Linux in interactive mode:

$ **rigctl -m 1014 -r /dev/ttyUSB1**

Start **rigctl** for a Yaesu FT-920 using COM1 on MS Windows while generating TRACE output to *stderr*:

> **rigctl -m 1014 -r COM1 -vvvvv**

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

$ **rigctl -m 1014 -r /dev/ttyUSB1 -s 4800 -C stop_bits=2**

Start **rigctl** for an Elecraft K3 using a USB to serial adapter while specifying a command terminator for the **w** command:

$ **rigctl -m 2029 -r /dev/ttyUSB0 -t';'**

Connect to a running **rigctld** with radio model 2 ("NET rigctl") on the local host and specifying the TCP port, setting frequency and mode:

$ **rigctl -m 2 -r localhost:4532 F 7253500 M LSB 0**

BUGS

set_chan has no entry method as of yet, hence left unimplemented.

This almost empty section...

Report bugs to:
Hamlib Developer mailing list

COPYING

SEE ALSO

less(1), **more**(1), **rigctld**(1), **hamlib**(7)

COLOPHON

Links to the Hamlib Wiki, Git repository, release archives, and daily snapshot archives are available via hamlib.org.