

Ham Radio Control Libraries

**API Reference for Version 1.1.0
(ALPHA)**

Nathan Bargmann

Ham Radio Control Libraries: API Reference for Version 1.1.0 (ALPHA)

by Nathan Bargmann

Published 23 September 2001

Copyright © 2000, 2001 by Frank Singleton, VK3FCS & KM5WS; Stephane Fillod, F4CFE;
Nate Bargmann, N0NB

Abstract

Demonstrate the need for Hamlib, document the Hamlib API, provide an introduction to writing a program using Hamlib, and provide an introduction to authoring a backend library to control a radio or other device.

Hamlib-doc - Ham Radio Control Libraries Documentation

Copyright (C) 2000-2001 Stephane Fillod, Frank Singleton, Nate Bargmann. This documentation file is part of the Hamlib package.

Hamlib-doc is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Hamlib-doc is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Hamlib-doc; see the file COPYING. If not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Revision History

Revision 0.1.1 23 September 2001 Revised by: nfb

Final final draft for 1.1.0 (ALPHA)

Revision 0.1.0 24 February 2001 Revised by: nfb

Final draft for 1.1.0 (ALPHA)

Revision 0.0.1 4 February 2001 Revised by: nfb

Rough draft of Preface and Chapter 1. Posted on the Web for critique

Table of Contents

Preface.....	i
1. Introduction to Hamlib.....	1
1.1. Overview of Hamlib.....	1
1.2. Hamlib is Free Software	1
1.3. Hamlib development resources.....	2
1.4. Getting Hamlib.....	2
1.4.1. Latest stable version.....	2
1.4.2. Latest development version.....	2
1.4.2.1. Accessing anonymous CVS.....	3
1.4.2.1.1. Anonymous CVS instructions.....	3
1.5. Building Hamlib	4
1.5.1. Unpacking the source archive	4
1.5.1.1. Using tar to extract the archive	4
1.5.2. Compiling Hamlib	5
1.6. Summary	6
2. hamlib API Reference.....	7
rigerror	7
foreach_opened_rig.....	9
rig_init	10
rig_open	11
rig_close.....	13
rig_cleanup.....	15
rig_set_freq	17
rig_get_freq.....	19
rig_set_mode.....	21
rig_get_mode	23
rig_set_vfo	25
rig_get_vfo.....	26
rig_set_ptt	27
rig_get_ptt.....	29
rig_set_rptr_shift.....	31
rig_get_rptr_shift	33
rig_set_rptr_offs.....	35
rig_get_rptr_offs	37
rig_set_split_freq	39
rig_get_split_freq.....	41
rig_set_split.....	43

rig_get_split	45
rig_set_rit	47
rig_get_rit.....	49
rig_set_ts	51
rig_get_ts.....	53
rig_power2mW	55
rig_mW2power	57
rig_set_ctcss.....	59
rig_get_ctcss	61
rig_set_dcs	63
rig_get_dcs.....	65
rig_set_ctcss_sql.....	67
rig_get_ctcss_sql.....	69
rig_set_dcs_sql.....	71
rig_get_dcs_sql	73
rig_set_poweron.....	75
rig_set_poweroff.....	76
rig_set_level	77
rig_get_level.....	79
rig_has_level	81
rig_has_set_level.....	83
rig_has_func.....	85
rig_set_func.....	87
rig_get_func	89
rig_set_mem.....	91
rig_get_mem	93
rig_mv_ctl.....	95
rig_set_bank.....	96
rig_set_channel	98
rig_get_channel.....	99
rig_get_range	100
rig_set_trn	101
rig_get_trn.....	103
rig_get_info.....	105

List of Examples

2-1.82
2-1.84
2-1.86

Preface

The Personal Computer has arguably become as prominent of a fixture in the modern hamshack as HF transceivers and handheld radios. The first PCs hams used combined with an interface also known as a terminal unit, TU, replaced the noisy mechanical radio teletype, RTTY, machines. Gradually, other uses became apparent for the hamshack PC such as logging contact information, operating newer digital modes such as AMTOR, a forward error correcting (FEC) mode of RTTY, and packet, an error checked way of transferring binary data between computers.

In the mid to late 1980s radio manufacturers began to offer computer control capability to their offerings. Now the shack PC had one more duty added to its repertoire. However, the amount of control over a given radio's capability by computer control was often much less than from the front panel. In large part was due to many analog controls on the front panel that weren't under control of the radio's CPU. Thus most radio control has been limited to control of frequency values of the variable frequency oscillators, VFO, and recalling and setting the radio's memory channels.

More recent models have more of the front panel controls under the supervision of the radio's CPU, consequently manufacturers have enabled more control of the radio's functions through the computer interface. This has created an explosion of command functions not just between the different manufacturers, but also among a given manufacturer's product line. There may also exist subtle differences in later versions of a given model.

For the software author wishing to incorporate even the most basic of radio control functions, the task is daunting. Even the long lived logging program CT™ by Ken Wolff, K1EA has reports of subtle bugs in one radio model or another (this is not intended as a knock of Ken or his effort, rather it is an observation of the complexity of supporting a multitude of radios in a monolithic program by one or a few authors). Is every programmer doomed to recreate radio control themselves with varying degrees of success? The answer is, no!

Enter the Hamlib project. Hamlib aims to provide application authors with a single application programming interface, API, regardless of the actual radio in use. Thus Hamlib is not an end user application, but is a middle software layer that acts as a translator between the API and the specific radio commands.

This manual will attempt to explain in detail the Hamlib API, how to use Hamlib for writing radio control software, how to write a radio backend library, and an overview of Hamlib's structure and design.

Finally, Hamlib is Free Software licensed under the GNU Public License, GPL, version

2. This will have certain implications if you are a software author considering using Hamlib as a part of your project. Consult the COPYING file in the base directory of the Hamlib source distribution for more details.

Chapter 1. Introduction to Hamlib

When browsing through the owners manual of that new radio, the pages devoted to the computer commands seem like an afterthought. While the manufacturers are not interested in becoming software houses, they do adequately document the computer control capabilities which allow independent control software to be written. With the myriad possibilities of radios and manufacturers available, writing that ultimate logging or PSK31 application can be a daunting task if even basic radio control support is desired. The Ham Radio Control Libraries project aims to develop a solution to this problem.

1.1. Overview of Hamlib

Hamlib itself is not an end-user application for radio control. Rather, it is a collection of libraries, both shared, or dynamic linked libraries, if you will, and static libraries that provide end-user applications with a common means of accessing and controlling radios (and perhaps other ham radio related peripheral devices in the future) directly connected to a computer or remotely over a network. Hamlib is currently being developed on the Linux operating system, but plans include it being usable on various flavors of UNIX™ and Windows™, or whatever platform GNU autoconf will support.

Hamlib consists of several parts. The application programming interface, API, shared library is `libhamlib-1.1.0.so` which is installed in `/usr/local/lib` by default. For ease of use when linking, `libhamlib.so` is provided as a symbolic link to the latest version of Hamlib installed. Of course, the installation directory may be changed by passing the proper option to the **configure** script in the base directory of the source distribution. While the static library is `libhamlib.a` and installed in `/usr/local/lib` as well.

The second main part of Hamlib consists of a number of "backend" libraries each able to communicate to a specific radio. For example, `libhamlib-ft747.so` is the shared backend library that provides Hamlib access to the Yaesu FT-747 radio. By default the backend libraries are also installed in `/usr/local/lib`. Both shared and static libraries are provided by the default installation.

1.2. Hamlib is Free Software

The Hamlib libraries are Free Software licensed under the GNU Public License, GPL,

version 2. It is important to be aware that use of Hamlib in a proprietary program has severe restrictions placed on it by the GPL. As a result one must carefully consider what kind of license to use for your program. Of course we encourage using the GPL for your program as it adds to the pool of available Free Software to the ham community.

The advantages of Free Software are multitude, but the primary ones include accessibility of your code to others who can fix problems or add new functionality. Another advantage is that your code is always available to be studied by other experimenters and your code has a much lower chance of becoming dead bits that can't be used on newer operating systems. For an experimenter's hobby like ham radio, Free Software offers many more advantages than disadvantages to you and the ham community.

1.3. Hamlib development resources

If you are interested in working on Hamlib development itself, there exist a few resources on the World Wide Web. The main project page is at <http://sourceforge.net/projects/hamlib/>. A homepage is currently in development at <http://hamlib.sourceforge.net> A development mailing list is hosted by <http://sourceforge.net>. Subscription information and an archive can be accessed through the Hamlib project main page.

1.4. Getting Hamlib

At this time Hamlib is not included as a binary package in any major distribution that we're aware of (hopefully this will change soon). Until then you may retrieve the source from the Hamlib project page at <http://sourceforge.net/projects/hamlib/>.

1.4.1. Latest stable version

The latest stable version is 1.1.0 (ALPHA). Currently the project is in its early stages and only a few backend libraries are included. Hamlib is currently in heavy development.

1.4.2. Latest development version

The latest development code is available via anonymous CVS through the project page (<http://sourceforge.net/projects/hamlib/>).

1.4.2.1. Accessing anonymous CVS

The following instructions are copied from the Sourceforge (<http://sourceforge.net>) website (modified with hamlib in the right places) and did work for me.

1.4.2.1.1. Anonymous CVS instructions

Hamlib's SourceForge CVS repository can be checked out through anonymous (pserver) CVS with the following instruction set. When prompted for a password for *anonymous*, simply press the **Enter** key.

```
myhost:~/src $ cvs -d:pserver:anonymous@cvs.hamlib.\
> sourceforge.net:/cvsroot/hamlib login
myhost:~/src $ cvs -z3 -d:pserver:anonymous@cvs.hamlib.\
> sourceforge.net:/cvsroot/hamlib co hamlib
```

Working with long commandlines: Long commands like those above are difficult to work with because once the line wraps the `Bash(1)` shell seems to start doing weird things. The trick is breaking the line into two (or more) parts with the “\” character. When the right edge of the screen is reached simply add \ to the end of the text you are typing and then press **Enter**. You will receive a > from `Bash(1)` and you may continue typing the command. If there is no space character in the command you are typing, be sure you don't add a space before the \ or at the beginning of the next line. If you break the line where a space would exist in the command, either putting the space before the \ or at the beginning of the next line. `Bash(1)` will splice the lines together to form one command once it receives a **Enter** character not preceded by a \.

Updates from within the hamlib directory do not need the **-d** parameter.

If you get the following error:

```
cvs login: failed to open /home/user/.cvspass for reading:
No such file or directory
cvs [login aborted]: fatal error: exiting
```

You can probably solve this by using the **touch** command to create the file `.cvspass` in your home directory:

```
myhost:~ $ touch .cvspass
```

1.5. Building Hamlib

Building Hamlib from source isn't as daunting as it may seem at first, thanks to GNU **autoconf**, a tool used by the developers that generates the **configure** script found in the base directory of the source distribution. Running **configure** will test your system to be sure that any required packages for building Hamlib are present. While **configure** checks for many components, the only critical dependency is that the C library development header files are installed. Of course, you'll need a C compiler and its associated libraries.

1.5.1. Unpacking the source archive

While my favorite method of unpacking `.tar.gz` files is to use the Linux version of the Swiss Army Knife, Midnight Commander, the instructions provided are for using **tar** at the command prompt.

1.5.1.1. Using tar to extract the archive

The first order of business is choosing a location for the source distribution. Some may choose to place the archive under `/usr/local/src`, or may prefer to work within their home directory. The disadvantage of working in `/usr/local/src` is that one must either be logged in as `root` or be a member of a group such as `staff` that has write permissions on the directory. The advantage of working in one's home directory is that writing and deleting files can be done with much lower risk of damage to the system areas of the filesystem. Either way, you will need to be logged in as `root` to install the libraries after compiling. On with unpacking the archive.

For this example I will make a few assumptions, the archive is downloaded and stored in `~/Download` and the source distribution will be installed in `~/src`.

Interpreting ~: If you are new to UNIX™ type systems, you may be puzzled just what `~` prepended to a path name means. It is simply a short hand for your home directory. If your user name is `fred`, then `~` refers to `/home/fred` on most systems, of course there are exceptions. If you are logged in as `root` then `~` refers to `/root`.

The following sequence of commands will get the Hamlib archive to the right place (substitute your paths in the examples). First we'll move the archive into the directory where it will be extracted then use the `tar` command to extract the archive into its own directory.

```
myhost:~ $ mv Download/hamlib-1.1.0.tar.gz src
myhost:~ $ cd src
myhost:~/src $ tar xvfz hamlib-1.1.0.tar.gz
```

Now you should have a directory called `hamlib-1.1.0` in the directory you executed the `tar` command. This would be a good time to familiarize yourself with the files in the archive.

1.5.2. Compiling Hamlib

Thanks to the clever design of GNU `autoconf` compiling Hamlib is as easy as running:

```
myhost:~/src/hamlib-1.1.0 $ ./configure
```

The `configure` script checks for the presence of the proper development files required to build Hamlib. After the checks `configure` then creates the Makefiles from the included templates in the archive. The next step is to compile Hamlib:

```
myhost:~/src/hamlib-1.1.0 $ make
```

Now there should be considerable output to the screen during the compile process. The main thing here is to make sure that `gcc` doesn't fail while reporting an error. The most

common failure is a message saying that a certain file cannot be found. Most likely the named file will have a .h extension which means the development files of a required library aren't installed on your system. As of this writing only the glibc development files are required.

1.6. Summary

Hamlib is a tool for software authors wishing to take advantage of the computer control capabilities of modern transceivers and other devices used around the radio shack. When Hamlib reaches maturity it will likely be available in your favorite packaging format and manually compiling it won't be necessary unless you wish to customize Hamlib itself.

The remainder of this manual assumes a working knowledge of UNIX™ type systems. If you are new to Linux, I suggest getting a copy of *Running Linux* by O'Reilly and Associates from your local bookstore. You can preview this excellent reference on the Web at <http://www.oreilly.com/catalog/runux3/>.

Chapter 2. hamlib API Reference

This chapter documents the Hamlib API for 1.1.0 (ALPHA). Each function of the API occurring in `rig.c` at the time of release is documented here. Every effort was made to ensure all arguments and return values are correct, but, as always, the source is definitive. Errors are bound to have occurred and others will be blamed!

rigerror

Name

`rigerror` — return string describing error code

Synopsis

```
const char * rigerror (int errnum);
```

Arguments

errnum

The error code

Description

The `rigerror` function returns a string describing the error code passed in the argument *errnum*.

RETURN VALUE

The `rigerror` function returns the appropriate description string, or an unknown error message if the error code is unknown.

TODO

check table bounds, support gettext

foreach_opened_rig

Name

`foreach_opened_rig` — execs `cfunc()` on each opened rig

Synopsis

```
int foreach_opened_rig (int (*cfunc) (RIG *, void *), void *  
data);
```

Arguments

cfunc

The function to be executed on each rig

data

Data pointer to be passed to *cfunc*

Description

The `foreach_opened_rig` function calls `cfunc` function for each opened rig. The contents of the opened rig table is processed in random order according to a function pointed to by *cfunc*, which is called with two arguments, the first pointing to the `&RIG` handle, the second to a data pointer *data*. If *data* is not needed, then it can be set to `NULL`. The processing of the opened rig table is stopped when `cfunc` returns 0.

RETURN VALUE

The `foreach_opened_rig` function always returns `RIG_OK`.

rig_init

Name

`rig_init` — allocate a new &RIG handle

Synopsis

```
RIG * rig_init (rig_model_t rig_model);
```

Arguments

rig_model

The rig model for this new handle

Description

The `rig_init` function allocates a new &RIG handle and initialize the associated data for *rig_model*.

RETURN VALUE

The `rig_init` function returns a pointer to the &RIG handle or `NULL` if memory allocation failed or *rig_model* is unknown.

SEE ALSO

`rig_cleanup` ([api-rig-cleanup.html](#)), `rig_open` ([api-rig-open.html](#))

rig_open

Name

`rig_open` — open the communication to the rig

Synopsis

```
int rig_open (RIG * rig);
```

Arguments

rig

The &RIG handle of the radio to be opened

Description

The `rig_open` function opens communication to a radio which &RIG handle has been passed by argument.

RETURN VALUE

The `rig_open` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

ERRORS

`RIG_EINVAL` *rig* is NULL or inconsistent. `RIG_ENIMPL` port type communication is not implemented yet.

SEE ALSO

`rig_close` ([api-rig-close.html](#))

rig_close

Name

`rig_close` — close the communication to the rig

Synopsis

```
int rig_close (RIG * rig);
```

Arguments

rig

The &RIG handle of the radio to be closed

Description

The `rig_close` function closes communication to a radio which &RIG handle has been passed by argument.

RETURN VALUE

The `rig_close` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

ERRORS

`RIG_EINVAL` *rig* is NULL or inconsistent.

SEE ALSO

`rig_open` ([api-rig-open.html](#)), `rig_cleanup` ([api-rig-cleanup.html](#))

rig_cleanup

Name

`rig_cleanup` — release a closed rig struct

Synopsis

```
int rig_cleanup (RIG * rig);
```

Arguments

rig

The rig handle

Description

The `rig_cleanup` function releases a rig struct which port has already been closed.

RETURN VALUE

The `rig_get_freq` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

ERRORS

`RIG_EINVAL` *rig* is NULL or inconsistent.

SEE ALSO

`rig_close` ([api-rig-close.html](#))

rig_set_freq

Name

`rig_set_freq` — set the frequency of the current VFO

Synopsis

```
int rig_set_freq (RIG * rig, vfo_t vfo, freq_t freq);
```

Arguments

rig

The rig handle

vfo

The target VFO

freq

The frequency to set to

Description

The `rig_set_freq` function sets the frequency of the current VFO.

RETURN VALUE

The `rig_set_freq` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_freq` ([api-rig-get-freq.html](#))

rig_get_freq

Name

`rig_get_freq` — get the frequency of the current VFO

Synopsis

```
int rig_get_freq (RIG * rig, vfo_t vfo, freq_t * freq);
```

Arguments

rig

The rig handle

vfo

The target VFO

freq

The location where to store the current frequency

Description

The `rig_get_freq` function retrieves the frequency of the current VFO.

RETURN VALUE

The `rig_get_freq` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_freq` ([api-rig-set-freq.html](#))

rig_set_mode

Name

`rig_set_mode` — set the mode of the current VFO

Synopsis

```
int rig_set_mode (RIG * rig, vfo_t vfo, rmode_t mode, pbwidth_t width);
```

Arguments

rig

The rig handle

vfo

The target VFO

mode

The mode to set to

width

The passband width to set to

Description

The `rig_set_mode` function sets the mode of the current VFO. As a beginning, the backend is free to ignore the *width* argument, however, it would be nice to at least honor the WFM case.

RETURN VALUE

The `rig_set_mode` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_mode` ([api-rig-get-mode.html](#))

rig_get_mode

Name

`rig_get_mode` — get the mode of the current VFO

Synopsis

```
int rig_get_mode (RIG * rig, vfo_t vfo, rmode_t * mode,  
pbwidth_t * width);
```

Arguments

rig

The rig handle

vfo

The target VFO

mode

The location where to store the current mode

width

The location where to store the current passband width

Description

The `rig_set_mode` function retrieves the mode of the current VFO. If the backend is unable to determine the width, it must return `RIG_PASSBAND_NORMAL` as a default.

RETURN VALUE

The `rig_get_mode` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_mode` ([api-rig-set-mode.html](#))

rig_set_vfo

Name

`rig_set_vfo` — set the current VFO

Synopsis

```
int rig_set_vfo (RIG * rig, vfo_t vfo);
```

Arguments

rig

The rig handle

vfo

The VFO to set to

Description

The `rig_set_vfo` function sets the current VFO.

RETURN VALUE

The `rig_set_vfo` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_vfo` ([api-rig-get-vfo.html](#))

rig_get_vfo

Name

`rig_get_vfo` — get the current VFO

Synopsis

```
int rig_get_vfo (RIG * rig, vfo_t * vfo);
```

Arguments

rig

The rig handle

vfo

The location where to store the current VFO

Description

The `rig_get_vfo` function retrieves the current VFO.

RETURN VALUE

The `rig_get_vfo` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_vfo` ([api-rig-set-vfo.html](#))

rig_set_ptt

Name

`rig_set_ptt` — set PTT on/off

Synopsis

```
int rig_set_ptt (RIG * rig, vfo_t vfo, ptt_t ptt);
```

Arguments

rig

The rig handle

vfo

The target VFO

ptt

The PTT status to set to

Description

The `rig_set_ptt` function sets “Push-To-Talk” on/off.

RETURN VALUE

The `rig_set_ptt` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_ptt` (<api-rig-get-ptt.html>)

rig_get_ptt

Name

`rig_get_ptt` — get the status of the PTT

Synopsis

```
int rig_get_ptt (RIG * rig, vfo_t vfo, ptt_t * ptt);
```

Arguments

rig

The rig handle

vfo

The target VFO

ptt

The location where to store the status of the PTT

Description

The `rig_get_ptt` function retrieves the status of PTT (are we on the air?).

RETURN VALUE

The `rig_get_ptt` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_ptt` ([api-rig-set-ptt.html](#))

rig_set_rptr_shift

Name

`rig_set_rptr_shift` — set the repeater shift

Synopsis

```
int rig_set_rptr_shift (RIG * rig, vfo_t vfo, rptr_shift_t  
rptr_shift);
```

Arguments

rig

The rig handle

vfo

The target VFO

rptr_shift

The repeater shift to set to

Description

The `rig_set_rptr_shift` function sets the current repeater shift.

RETURN VALUE

The `rig_rptr_shift` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_rptr_shift` (<api-rig-get-rptr-shift.html>)

rig_get_rptr_shift

Name

`rig_get_rptr_shift` — get the current repeater shift

Synopsis

```
int rig_get_rptr_shift (RIG * rig, vfo_t vfo, rptr_shift_t *  
rptr_shift);
```

Arguments

rig

The rig handle

vfo

The target VFO

rptr_shift

The location where to store the current repeater shift

Description

The `rig_get_rptr_shift` function retrieves the current repeater shift.

RETURN VALUE

The `rig_get_rptr_shift` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_rptr_shift` (api-rig-set-rptr-shift.html)

rig_set_rptr_offs

Name

`rig_set_rptr_offs` — set the repeater offset

Synopsis

```
int rig_set_rptr_offs (RIG * rig, vfo_t vfo, shortfreq_t  
rptr_offs);
```

Arguments

rig

The rig handle

vfo

The target VFO

rptr_offs

The VFO to set to

Description

The `rig_set_rptr_offs` function sets the current repeater offset.

RETURN VALUE

The `rig_set_rptr_offs` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_rptr_offs` (api-rig-get-rptr-offs.html)

rig_get_rptr_offs

Name

`rig_get_rptr_offs` — get the current repeater offset

Synopsis

```
int rig_get_rptr_offs (RIG * rig, vfo_t vfo, shortfreq_t *  
rptr_offs);
```

Arguments

rig

The rig handle

vfo

The target VFO

rptr_offs

The location where to store the current repeater offset

Description

The `rig_get_rptr_offs` function retrieves the current repeater offset.

RETURN VALUE

The `rig_get_rptr_offs` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_rptr_offs` ([api-rig-set-rptr-offs.html](#))

rig_set_split_freq

Name

`rig_set_split_freq` — set the split frequencies

Synopsis

```
int rig_set_split_freq (RIG * rig, vfo_t vfo, freq_t rx_freq,  
freq_t tx_freq);
```

Arguments

rig

The rig handle

vfo

The target VFO

rx_freq

The receive split frequency to set to

tx_freq

The transmit split frequency to set to

Description

The `rig_set_split_freq` function sets the split frequencies.

RETURN VALUE

The `rig_set_split_freq` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_split_freq` ([api-rig-get-split-freq.html](#))

rig_get_split_freq

Name

`rig_get_split_freq` — get the current split frequencies

Synopsis

```
int rig_get_split_freq (RIG * rig, vfo_t vfo, freq_t * rx_freq,  
freq_t * tx_freq);
```

Arguments

rig

The rig handle

vfo

The target VFO

rx_freq

The location where to store the current receive split frequency

tx_freq

The location where to store the current receive split frequency

Description

The `rig_get_split_freq` function retrieves the current split frequencies.

RETURN VALUE

The `rig_get_split_freq` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_split_freq` ([api-rig-set-split-freq.html](#))

rig_set_split

Name

`rig_set_split` — set the split mode

Synopsis

```
int rig_set_split (RIG * rig, vfo_t vfo, split_t split);
```

Arguments

rig

The rig handle

vfo

The target VFO

split

The split mode to set to

Description

The `rig_set_split` function sets the current split mode.

RETURN VALUE

The `rig_set_split` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_split` (api-rig-get-split.html)

rig_get_split

Name

`rig_get_split` — get the current split mode

Synopsis

```
int rig_get_split (RIG * rig, vfo_t vfo, split_t * split);
```

Arguments

rig

The rig handle

vfo

The target VFO

split

The location where to store the current split mode

Description

The `rig_get_split` function retrieves the current split mode.

RETURN VALUE

The `rig_get_split` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_split` ([api-rig-set-split.html](#))

rig_set_rit

Name

`rig_set_rit` — set the RIT

Synopsis

```
int rig_set_rit (RIG * rig, vfo_t vfo, shortfreq_t rit);
```

Arguments

rig

The rig handle

vfo

The target VFO

rit

The RIT offset to adjust to

Description

The `rig_set_rit` function sets the current RIT offset. A value of 0 for *rit* disables RIT.

RETURN VALUE

The `rig_set_rit` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_rit` (<api-rig-get-rit.html>)

rig_get_rit

Name

`rig_get_rit` — get the current RIT offset

Synopsis

```
int rig_get_rit (RIG * rig, vfo_t vfo, shortfreq_t * rit);
```

Arguments

rig

The rig handle

vfo

The target VFO

rit

The location where to store the current RIT offset

Description

The `rig_get_rit` function retrieves the current RIT offset.

RETURN VALUE

The `rig_get_rit` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_rit` ([api-rig-set-rit.html](#))

rig_set_ts

Name

`rig_set_ts` — set the Tuning Step

Synopsis

```
int rig_set_ts (RIG * rig, vfo_t vfo, shortfreq_t ts);
```

Arguments

rig

The rig handle

vfo

The target VFO

ts

The tuning step to set to

Description

The `rig_set_ts` function sets the Tuning Step.

RETURN VALUE

The `rig_set_ts` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_ts` ([api-rig-get-ts.html](#))

rig_get_ts

Name

`rig_get_ts` — get the current Tuning Step

Synopsis

```
int rig_get_ts (RIG * rig, vfo_t vfo, shortfreq_t * ts);
```

Arguments

rig

The rig handle

vfo

The target VFO

ts

The location where to store the current tuning step

Description

The `rig_get_ts` function retrieves the current tuning step.

RETURN VALUE

The `rig_get_ts` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_ts` ([api-rig-set-ts.html](#))

rig_power2mW

Name

`rig_power2mW` — conversion utility from relative range to absolute in mW

Synopsis

```
int rig_power2mW (RIG * rig, unsigned int * mwpower, float  
power, freq_t freq, rmode_t mode);
```

Arguments

rig

The rig handle

mwpower

The location where to store the converted power in mW

power

The relative power

freq

The frequency where the conversion should take place

mode

The mode where the conversion should take place

Description

The `rig_power2mW` function converts a power value expressed in a range on a [0.0 .. 1.0] relative scale to the real transmit power in milli Watts the radio would emit. The

rig_power2mW

freq and *mode* where the conversion should take place must be also provided since the relative power is peculiar to a specific freq and mode range of the radio.

RETURN VALUE

The `rig_power2mW` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_mW2power` ([api-rig-mw2power.html](#))

rig_mW2power

Name

`rig_mW2power` — conversion utility from absolute in mW to relative range

Synopsis

```
int rig_mW2power (RIG * rig, float * power, unsigned int  
mwpower, freq_t freq, rmode_t mode);
```

Arguments

rig

The rig handle

power

The location where to store the converted relative power

mwpower

The power in mW

freq

The frequency where the conversion should take place

mode

The mode where the conversion should take place

Description

The `rig_mW2power` function converts a power value expressed in the real transmit power in milli Watts the radio would emit to a range on a [0.0 .. 1.0] relative scale. The

freq and *mode* where the conversion should take place must be also provided since the relative power is peculiar to a specific *freq* and *mode* range of the radio.

RETURN VALUE

The `rig_mW2power` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_power2mW` ([api-rig-power2mw.html](#))

rig_set_ctcss

Name

`rig_set_ctcss` — set CTCSS

Synopsis

```
int rig_set_ctcss (RIG * rig, vfo_t vfo, unsigned int tone);
```

Arguments

rig

The rig handle

vfo

The target VFO

tone

The tone to set to

Description

The `rig_set_ctcss` function sets the current Continuous Tone Controlled Squelch System (CTCSS) sub-audible tone. NB, *tone* is NOT in Hz, but in tenth of Hz! This way, if you want to set subaudible tone of 88.5 Hz for example, then pass 885 to this function. Also, to disable Tone squelch, set *tone* to 0.

RETURN VALUE

The `rig_set_ctcss` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, *cause* is set appropriately).

SEE ALSO

`rig_get_ctcss` ([api-rig-get-ctcss.html](#)), `rig_set_dcs` ([api-rig-set-dcs.html](#)),
`rig_get_dcs` ([api-rig-get-dcs.html](#))

rig_get_ctcss

Name

`rig_get_ctcss` — get the current CTCSS

Synopsis

```
int rig_get_ctcss (RIG * rig, vfo_t vfo, unsigned int * tone);
```

Arguments

rig

The rig handle

vfo

The target VFO

tone

The location where to store the current tone

Description

The `rig_get_ctcss` function retrieves the current Continuous Tone Controlled Squelch System (CTCSS) sub-audible tone. NB, *tone* is NOT in Hz, but in tenth of Hz! This way, if the function `rig_get_ctcss` returns a subaudible tone of 885 for example, then the real tone is 88.5 Hz. Also, a value of 0 for *tone* means the Tone squelch is disabled.

RETURN VALUE

The `rig_get_ctcss` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_ctcss` ([api-rig-set-ctcss.html](#)), `rig_set_dcs` ([api-rig-set-dcs.html](#)),
`rig_get_dcs` ([api-rig-get-dcs.html](#))

rig_set_dcs

Name

`rig_set_dcs` — set the current DCS

Synopsis

```
int rig_set_dcs (RIG * rig, vfo_t vfo, unsigned int code);
```

Arguments

rig

The rig handle

vfo

The target VFO

code

The tone to set to

Description

The `rig_set_dcs` function sets the current Digitally-Coded Squelch code.

RETURN VALUE

The `rig_set_dcs` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_dcs` ([api-rig-get-dcs.html](#)), `rig_set_ctcss` ([api-rig-set-ctcss.html](#)),
`rig_get_ctcss` ([api-rig-get-ctcss.html](#))

rig_get_dcs

Name

`rig_get_dcs` — get the current DCS

Synopsis

```
int rig_get_dcs (RIG * rig, vfo_t vfo, unsigned int * code);
```

Arguments

rig

The rig handle

vfo

The target VFO

code

The location where to store the current tone

Description

The `rig_get_dcs` function retrieves the current Digitally-Coded Squelch.

RETURN VALUE

The `rig_get_dcs` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_dcs` ([api-rig-get-dcs.html](#)), `rig_set_ctcss` ([api-rig-set-ctcss.html](#)),
`rig_get_ctcss` ([api-rig-get-ctcss.html](#))

rig_set_ctcss_sql

Name

`rig_set_ctcss_sql` — set CTCSS squelch

Synopsis

```
int rig_set_ctcss_sql (RIG * rig, vfo_t vfo, unsigned int tone);
```

Arguments

rig

The rig handle

vfo

The target VFO

tone

The PL tone to set the squelch to

Description

The `rig_set_ctcss_sql` function sets the current Continuous Tone Controlled Squelch System (CTCSS) sub-audible squelch tone. NB, *tone* is NOT in Hz, but in tenth of Hz! This way, if you want to set subaudible tone of 88.5 Hz for example, then pass 885 to this function. Also, to disable Tone squelch, set *tone* to 0.

RETURN VALUE

The `rig_set_ctcss_sql` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set

appropriately).

SEE ALSO

`rig_get_ctcss_sql` ([api-rig-get-ctcss-sql.html](#)), `rig_set_ctcss` ([api-rig-set-ctcss.html](#))

rig_get_ctcss_sql

Name

`rig_get_ctcss_sql` — get the current CTCSS squelch

Synopsis

```
int rig_get_ctcss_sql (RIG * rig, vfo_t vfo, unsigned int *  
tone);
```

Arguments

rig

The rig handle

vfo

The target VFO

tone

The location where to store the current tone

Description

The `rig_get_ctcss_sql` function retrieves the current Continuous Tone Controlled Squelch System (CTCSS) sub-audible squelch tone. NB, *tone* is NOT in Hz, but in tenth of Hz! This way, if the function `rig_get_ctcss` returns a subaudible tone of 885 for example, then the real tone is 88.5 Hz. Also, a value of 0 for *tone* means the Tone squelch is disabled.

RETURN VALUE

The `rig_get_ctcss_sql` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_ctcss_sql` ([api-rig-set-ctcss-sql.html](#)), `rig_get_ctcss` ([api-rig-get-ctcss.html](#))

rig_set_dcs_sql

Name

`rig_set_dcs_sql` — set the current DCS

Synopsis

```
int rig_set_dcs_sql (RIG * rig, vfo_t vfo, unsigned int code);
```

Arguments

rig

The rig handle

vfo

The target VFO

code

The tone to set to

Description

The `rig_set_dcs_sql` function sets the current Digitally-Coded Squelch code.

RETURN VALUE

The `rig_set_dcs_sql` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_dcs_sql` ([api-rig-get-dcs-sql.html](#)), `rig_set_dcs` ([api-rig-set-dcs.html](#))

rig_get_dcs_sql

Name

`rig_get_dcs_sql` — get the current DCS

Synopsis

```
int rig_get_dcs_sql (RIG * rig, vfo_t vfo, unsigned int * code);
```

Arguments

rig

The rig handle

vfo

The target VFO

code

The location where to store the current tone

Description

The `rig_get_dcs_sql` function retrieves the current Digitally-Coded Squelch.

RETURN VALUE

The `rig_get_dcs_sql` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_dcs_sql` ([api-rig-get-dcs.html](#)), `rig_get_dcs` ([api-rig-get-dcs.html](#))

rig_set_poweron

Name

`rig_set_poweron` — turn on the radio

Synopsis

```
int rig_set_poweron (RIG * rig);
```

Arguments

rig

The rig handle

Description

The `rig_set_poweron` function turns on the radio.

RETURN VALUE

The `rig_set_poweron` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_poweroff` ([api-rig-set-poweroff.html](#))

rig_set_poweroff

Name

`rig_set_poweroff` — turn off the radio

Synopsis

```
int rig_set_poweroff (RIG * rig);
```

Arguments

rig

The rig handle

Description

The `rig_set_poweroff` function turns off the radio.

RETURN VALUE

The `rig_set_poweroff` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_poweron` ([api-rig-set-poweron.html](#))

rig_set_level

Name

`rig_set_level` — set a radio level setting

Synopsis

```
int rig_set_level (RIG * rig, vfo_t vfo, setting_t level,  
value_t val);
```

Arguments

rig

The rig handle

vfo

The target VFO

level

The level setting

val

The value to set the level setting to

Description

The `rig_set_level` function sets the level of a setting. The level value *val* can be a float or an integer. See `&value_t` for more information.

RETURN VALUE

The `rig_set_level` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_has_set_level` ([api-rig-has-set-level.html](#)), `rig_get_level` ([api-rig-get-level.html](#))

rig_get_level

Name

`rig_get_level` — get the level of a level

Synopsis

```
int rig_get_level (RIG * rig, vfo_t vfo, setting_t level,  
value_t * val);
```

Arguments

rig

The rig handle

vfo

The target VFO

level

The level setting

val

The location where to store the value of *level*

Description

The `rig_get_level` function retrieves the value of a *level*. The level value *val* can be a float or an integer. See `&value_t` for more information.

RETURN VALUE

The `rig_get_level` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_has_level` ([api-rig-has-level.html](#)), `rig_set_level` ([api-rig-set-level.html](#))

rig_has_level

Name

`rig_has_level` — check retrieval ability of level settings

Synopsis

```
setting_t rig_has_level (RIG * rig, setting_t level);
```

Arguments

rig

The rig handle

level

The level settings

Description

The `rig_has_level` “macro” checks if a rig can *get* a level setting. Since the *level* is a OR’ed bitwise argument, more than one level can be checked at the same time.

RETURN VALUE

The `rig_has_level` “macro” returns a bit wise mask of supported level settings that can be retrieve, 0 if none supported.

EXAMPLE

Example 2-1.

```
if (rig_has_level(my_rig, RIG_LVL_STRENGTH)) disp_Smeter;
```

SEE ALSO

`rig_has_set_level` ([api-rig-has-set-level.html](#)), `rig_get_level` ([api-rig-get-level.html](#))

rig_has_set_level

Name

`rig_has_set_level` — check settable ability of level settings

Synopsis

```
setting_t rig_has_set_level (RIG * rig, setting_t level);
```

Arguments

rig

The rig handle

level

The level settings

Description

The `rig_has_set_level` “macro” checks if a rig can *set* a level setting. Since the *level* is a OR’ed bitwise argument, more than one level can be check at the same time.

RETURN VALUE

The `rig_has_set_level` “macro” returns a bit wise mask of supported level settings that can be set, 0 if none supported.

EXAMPLE

Example 2-1.

```
if (rig_has_set_level(my_rig, RIG_LVL_RFPOWER)) crank_tx;
```

SEE ALSO

`rig_has_level` ([api-rig-has-level.html](#)), `rig_set_level` ([api-rig-set-level.html](#))

rig_has_func

Name

`rig_has_func` — check ability of radio functions

Synopsis

```
setting_t rig_has_func (RIG * rig, setting_t func);
```

Arguments

rig

The rig handle

func

The functions

Description

The `rig_has_func` “macro” checks if a rig supports a set of functions. Since the *func* is a OR’ed bitwise argument, more than one function can be checked at the same time.

RETURN VALUE

The `rig_has_func` “macro” returns a bit wise mask of supported functions, 0 if none supported.

EXAMPLE

Example 2-1.

```
if (rig_has_func(my_rig, RIG_FUNC_FAGC)) disp_fagc_button;
```

SEE ALSO

`rig_set_func` ([api-rig-set-func.html](#)), `rig_get_func` ([api-rig-get-func.html](#))

rig_set_func

Name

`rig_set_func` — activate/desactivate functions of radio

Synopsis

```
int rig_set_func (RIG * rig, vfo_t vfo, setting_t func, int status);
```

Arguments

rig

The rig handle

vfo

The target VFO

func

The functions to activate

status

The status (on or off) to set to

Description

The `rig_set_func` function activate/desactivate functions of the radio.

RETURN VALUE

The `rig_set_func` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_func` ([api-rig-get-func.html](#))

rig_get_func

Name

`rig_get_func` — get the status of functions of the radio

Synopsis

```
int rig_get_func (RIG * rig, vfo_t vfo, setting_t * func);
```

Arguments

rig

The rig handle

vfo

The target VFO

func

The location where to store the function status

Description

The `rig_get_func` function retrieves the status of functions of the radio. Only the function bits set to 1 will be queried. On return, *func* will hold the status of functions (bit set to 1 = activated, bit set to 0 = deactivated).

RETURN VALUE

The `rig_get_func` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_func` ([api-rig-set-func.html](#))

rig_set_mem

Name

`rig_set_mem` — set the current memory channel number

Synopsis

```
int rig_set_mem (RIG * rig, vfo_t vfo, int ch);
```

Arguments

rig

The rig handle

vfo

The target VFO

ch

The memory channel number

Description

The `rig_set_mem` function sets the current memory channel number. It is not mandatory for the radio to be in memory mode. Actually it depends on rigs. YMMV.

RETURN VALUE

The `rig_set_mem` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_get_mem` ([api-rig-get-mem.html](#))

rig_get_mem

Name

`rig_get_mem` — get the current memory channel number

Synopsis

```
int rig_get_mem (RIG * rig, vfo_t vfo, int * ch);
```

Arguments

rig

The rig handle

vfo

The target VFO

ch

The location where to store the current memory channel number

Description

The `rig_get_mem` function retrieves the current memory channel number. It is not mandatory for the radio to be in memory mode. Actually it depends on rigs. YMMV.

RETURN VALUE

The `rig_get_mem` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_mem` ([api-rig-set-mem.html](#))

rig_mv_ctl

Name

`rig_mv_ctl` — perform Memory/VFO operations

Synopsis

```
int rig_mv_ctl (RIG * rig, vfo_t vfo, mv_op_t op);
```

Arguments

rig

The rig handle

vfo

The target VFO

op

The Memory/VFO operation to perform

Description

The `rig_mv_ctl` function performs Memory/VFO operation. See `&mv_op_t` for more information.

RETURN VALUE

The `rig_mv_ctl` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

rig_set_bank

Name

`rig_set_bank` — set the current memory bank

Synopsis

```
int rig_set_bank (RIG * rig, vfo_t vfo, int bank);
```

Arguments

rig

The rig handle

vfo

The target VFO

bank

The memory bank

Description

The `rig_set_bank` function sets the current memory bank. It is not mandatory for the radio to be in memory mode. Actually it depends on rigs. YMMV.

RETURN VALUE

The `rig_set_bank` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_mem` ([api-rig-set-mem.html](#))

rig_set_channel

Name

`rig_set_channel` — set channel data

Synopsis

```
int rig_set_channel (RIG * rig, const channel_t * chan);
```

Arguments

rig

The rig handle

chan

The location of data to set for this channel

Description

The `rig_set_channel` function sets the data associated with a channel. See `&channel_t` for more information.

RETURN VALUE

The `rig_set_channel` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_channel` ([api-rig-get-channel.html](#))

rig_get_channel

Name

`rig_get_channel` — get channel data

Synopsis

```
int rig_get_channel (RIG * rig, channel_t * chan);
```

Arguments

rig

The rig handle

chan

The location where to store the channel data

Description

The `rig_get_channel` function retrieves the data associated with the channel `chan->channel_num`. See `&channel_t` for more information.

RETURN VALUE

The `rig_get_channel` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_set_channel` ([api-rig-set-channel.html](#))

rig_get_range

Name

`rig_get_range` — find the `freq_range` of `freq/mode`

Synopsis

```
const freq_range_t * rig_get_range (const freq_range_t *  
range_list, freq_t freq, rmode_t mode);
```

Arguments

range_list

The range list to search from

freq

The frequency that will be part of this range

mode

The mode that will be part of this range

Description

The `rig_get_range` function returns the location of the `&freq_range_t` including `freq` and `mode`. Works for rx and tx range list as well.

RETURN VALUE

The `rig_get_range` function returns the location of the `&freq_range_t` if found, `NULL` if not found or if `range_list` is invalid.

rig_set_trn

Name

`rig_set_trn` — control the transceive mode

Synopsis

```
int rig_set_trn (RIG * rig, vfo_t vfo, int trn);
```

Arguments

rig

The rig handle

vfo

The target VFO

trn

The transceive status to set to

Description

The `rig_set_trn` function enable/disable the transceive handling of a rig and kick off async mode.

RETURN VALUE

The `rig_set_trn` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, `cause` is set appropriately).

SEE ALSO

`rig_get_trn` (<api-rig-get-trn.html>)

rig_get_trn

Name

`rig_get_trn` — get the current transceive mode

Synopsis

```
int rig_get_trn (RIG * rig, vfo_t vfo, int * trn);
```

Arguments

rig

The rig handle

vfo

The target VFO

trn

The location where to store the current transceive mode

Description

The `rig_get_trn` function retrieves the current status of the transceive mode, ie. if radio sends new status automatically when some changes happened on the radio.

RETURN VALUE

The `rig_get_trn` function returns `RIG_OK` if the operation has been successful, or a negative value if an error occurred (in which case, cause is set appropriately).

SEE ALSO

`rig_set_trn` ([api-rig-set-trn.html](#))

rig_get_info

Name

`rig_get_info` — get general information from the radio

Synopsis

```
unsigned char* rig_get_info (RIG * rig);
```

Arguments

rig

The rig handle

Description

The `rig_get_info` function retrieves some general information from the radio. This can include firmware revision, exact model name, or just nothing.

RETURN VALUE

The `rig_get_info` function returns a pointer to freshly allocated memory containing the ASCII string if the operation has been successful, or NULL if an error occurred.