WSA4000 CLI (Command Line Interface) Program Documentation

Generated by Doxygen 1.7.4

Thu Sep 8 2011 15:58:59

Contents

1 Introduction

This documentation, compiled using Doxygen, shows in details the code structure of the CLI (Command Line Interface) tool. It provides information on all the libraries involved.

The following diagram illustrates the different layers and libraries involved in interfacing with a WSA on the PC side.

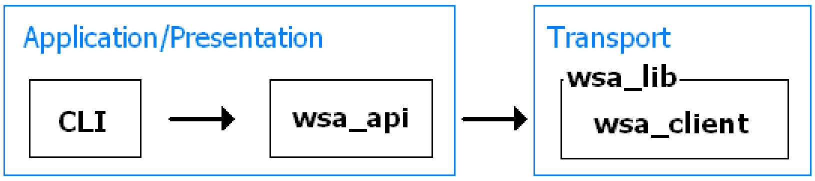


Figure 1: Interface Layers to WSA on PC Side

The CLI interfaces to a WSA through the wsa_api library, which provides functions to set/get particular settings or data from the WSA. The wsa_api encodes the commands into SCPI syntax scripts, which are sent to a WSA through the wsa_lib library. Subsequently decodes any responses or packet coming back from the WSA through the wsa_lib.

The wsa_lib, thus, is the main gateway to a WSA box from a PC. The wsa_lib has functions to open, close, send/receive commands, querry the WSA box status, and get data. In this CLI version, wsa_lib calls the wsa_client's functions in the transport layer to establish TCP/IP specific connections. Other connection methods such as USB could be added to the transport layer later on. The wsa_lib, thus, abstracts away the interface method from any application/presentation program calling it.

The CLI, hence, is a direct example of how the wsa_api library could be used. VRT data packet will be decoded before saving into a file.

The WSA4000 CLI is designed using mixed C/C++ languages. The CLI when executed will run in a Windows command prompt console. List of commands available with the CLI is listed in the print_cli_menu() function.

1.1 Limitations in v1.0

The following features are not yet supported with the CLI:

- DC correction. Need Nikhil to clarify on that.
- IQ correction. Same as above.
- Automatic finding of a WSA box(s) on a network.
- Set sample sizes. 1024 size for now.
- Triggers.
- Gain calibrarion. TBD with triggers.
- USB interface method - might never be available.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 wsa_descriptor Struct Reference

This structure stores WSA information.

Data Fields

- char prod_name [50]
- char prod_serial [20]

- char prod_version [20]
- char rfe_name [50]
- char rfe_version [20]
- char fw_version [20]
- char intf_type [20]
- uint64_t inst_bw
- uint64_t max_sample_size
- uint64_t max_tune_freq
- uint64_t min_tune_freq
- uint64_t freq_resolution
- float max_if_gain
- float min_if_gain
- float abs_max_amp [NUM_RF_GAINS]

4.1.1 Field Documentation

4.1.1.1 float abs_max_amp

An array storing the absolute maximum RF input level in dBm for each RF gain setting of the RFE use. Operating a WSA device at these absolute maximums may cause damage to the device.

4.1.1.2 uint64_t freq_resolution

The frequency resolution in Hz that a WSA's centre frequency can be incremented.

4.1.1.3 char fw_version

The firmware version currently in the WSA.

4.1.1.4 uint64_t inst_bw

The WSA instantaneous bandwidth in Hz.

4.1.1.5 char intf_type

The interface method to a WSA. Available: "TCPIP" ("USB" TBD).

4.1.1.6 float max_if_gain

The maximum IF gain in dB that a WSA's RFE can be set.

4.1.1.7 uint64_t max_sample_size

The maximum number of continuous I and Q data samples the WSA can capture per frame.

4.1.1.8 uint64_t max_tune_freq

The maximum frequency in Hz that a WSA's RFE can be tuned to.

4.1.1.9 float min_if_gain

The minimum IF gain in dB that a WSA's RFE can be set.

4.1.1.10 uint64_t min_tune_freq

The minimum frequency in Hz that a WSA's RFE can be tuned to.

4.1.1.11 char prod_name

WSA product name.

4.1.1.12 char prod_serial

WSA product serial number.

4.1.1.13 char prod_version

WSA product version number.

4.1.1.14 char rfe_name

WSA product name.

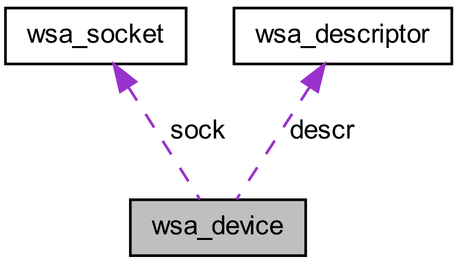
4.1.1.15 char rfe_version

WSA product version number.

4.2 wsa_device Struct Reference

A structure containing the components associate with each WSA device.

Collaboration diagram for wsa_device:



Data Fields

- struct wsa_descriptor descr
- struct wsa_socket sock

4.2.1 Field Documentation

4.2.1.1 struct wsa_descriptor descr

The information component of the WSA, stored in wsa_descriptor.

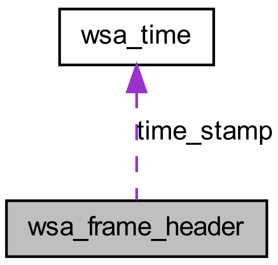
4.2.1.2 struct wsa_socket sock

The socket structure component of the WSA, used for TCPIP connection.

4.3 wsa_frame_header Struct Reference

This structure contains header information related to each frame read by wsa_get_-frame().

Collaboration diagram for wsa_frame_header:



Data Fields

- char prod_serial [20]
- uint64_t freq
- char gain [10]
- uint32_t sample_size
- struct wsa_time time_stamp

4.3.1 Field Documentation

4.3.1.1 uint64_t freq

The center frequency (Hz) to which the RF PLL is tuned.

4.3.1.2 char gain

The amplification in the radio front end at the time a WSA data frame is captured.

4.3.1.3 char prod_serial

WSA product version number.

4.3.1.4 uint32_t sample_size

Number of {I, Q} samples pairs per WSA data frame.

4.3.1.5 struct wsa_time time_stamp

The time when a data frame capture begins, stored in wsa_time structure.

4.4 wsa_resp Struct Reference

This structure contains the response information for each query.

Data Fields

- int64_t status
- char result [MAX_STR_LEN]

4.4.1 Field Documentation

4.4.1.1 char result

The resulted string responded to a query.

4.4.1.2 int32_t status

The status of the query. Positive number when success, negative when failed.

4.5 wsa_socket Struct Reference

A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition.

Data Fields

- SOCKET cmd
- SOCKET data

4.5.1 Field Documentation

4.5.1.1 SOCKET cmd

The command socket for command controls and queries. The string protocol used for this socket is HISLIP.

4.5.1.2 SOCKET data

The data socket used for streaming of data

4.6 wsa_time Struct Reference

This structure contains the time information. It is used for the time stamp in a frame header.

Data Fields

- int32_t sec
- uint32_t nsec

4.6.1 Field Documentation

4.6.1.1 int32_t nsec

Nanoseconds after the second (0 - 999 999 999).

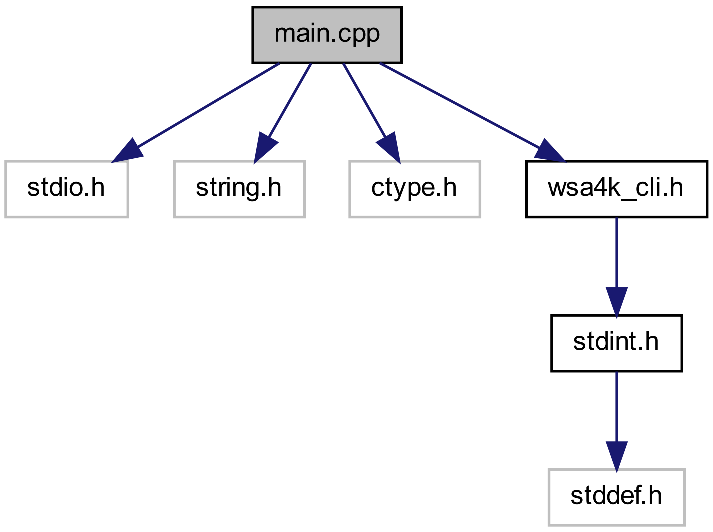
4.6.1.2 int32_t sec

The number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC.

5 File Documentation

5.1 main.cpp File Reference

Include dependency graph for main.cpp:



Functions

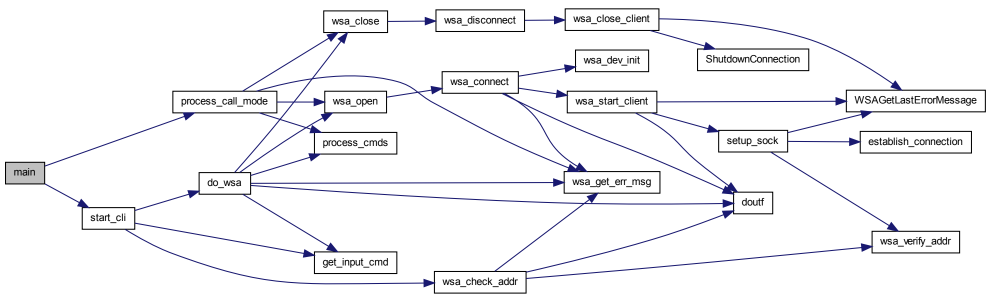
- int32_t main (int32_t argc, char ∗argv[])

5.1.1 Function Documentation

5.1.1.1 int32_t main (int32_t *argc*, char ∗ *argv[]*)

Starting point

Here is the call graph for this function:



5.2 ReadMe.txt File Reference

Variables

- and information about the platforms
- and information about the configurations
- and information about the and project features selected with the Application Wizard wsa4000_cli cpp This is the main application source file Other standard files

5.2.1 Variable Documentation

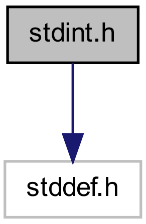
5.2.1.1 and information about the configurations

5.2.1.2 and information about the and project features selected with the Application Wizard wsa4000_cli cpp This is the main application source file Other standard files

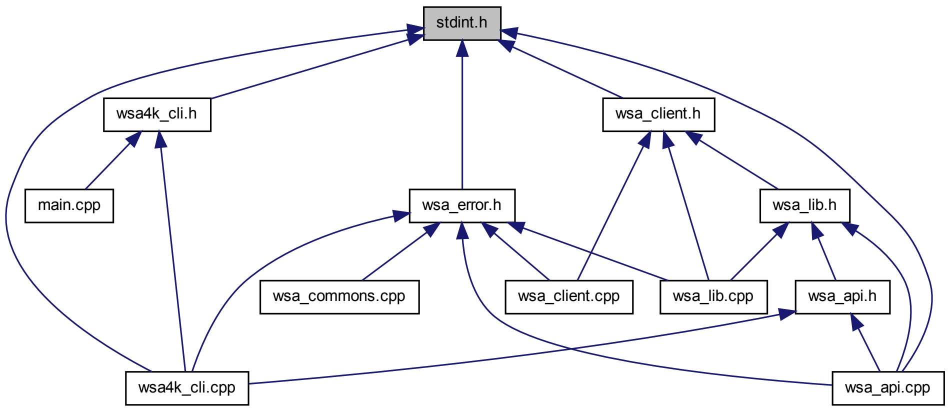
5.2.1.3 and information about the platforms

5.3 stdint.h File Reference

Include dependency graph for stdint.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef signed char int8_t
- typedef unsigned char uint8_t
- typedef short int16_t
- typedef unsigned short uint16_t
- typedef int int32_t
- typedef unsigned int uint32_t

- typedef long long int64_t
- typedef unsigned long long uint64_t

5.3.1 Typedef Documentation

5.3.1.1 typedef short **int16_t**

5.3.1.2 typedef int **int32_t**

5.3.1.3 typedef long long **int64_t**

5.3.1.4 typedef signed char **int8_t**

Exact-width integer types

5.3.1.5 typedef unsigned short **uint16_t**

5.3.1.6 typedef unsigned int **uint32_t**

5.3.1.7 typedef unsigned long long **uint64_t**

5.3.1.8 typedef unsigned char **uint8_t**

5.4 targetver.h File Reference

Defines

- #define _WIN32_WINNT 0x0600

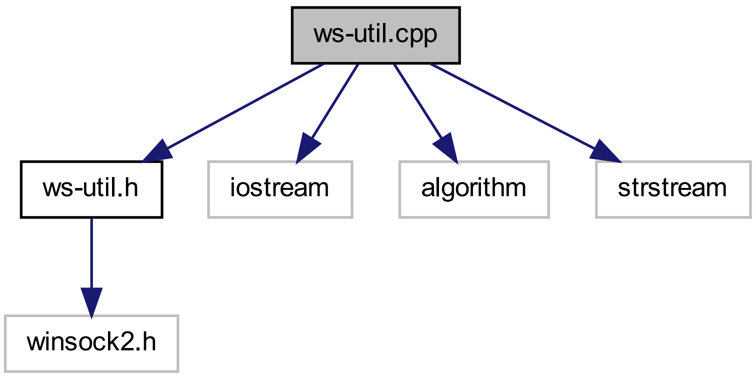
5.4.1 Define Documentation

5.4.1.1 #define _WIN32_WINNT 0x0600

5.5 test scpi.txt File Reference

5.6 ws-util.cpp File Reference

Include dependency graph for ws-util.cpp:



Data Structures

- struct **ErrorEntry**

Functions

- const char ∗ WSAGetLastErrorMessage (const char ∗pcMessagePrefix, int nErrorID)
- bool ShutdownConnection (SOCKET sd, char ∗sock_name)

Variables

- const int kBufferSize = 1024
- const int kNumMessages = sizeof(gaErrorList) / sizeof(ErrorEntry)

5.6.1 Function Documentation

5.6.1.1 bool ShutdownConnection (SOCKET *sd*, char ∗ *sock_name*)

5.6.1.2 const char∗ WSAGetLastErrorMessage (const char ∗ *pcMessagePrefix*, int *nErrorID*)

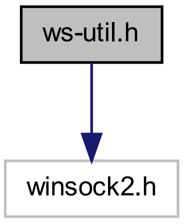
5.6.2 Variable Documentation

5.6.2.1 const int **kBufferSize** = 1024

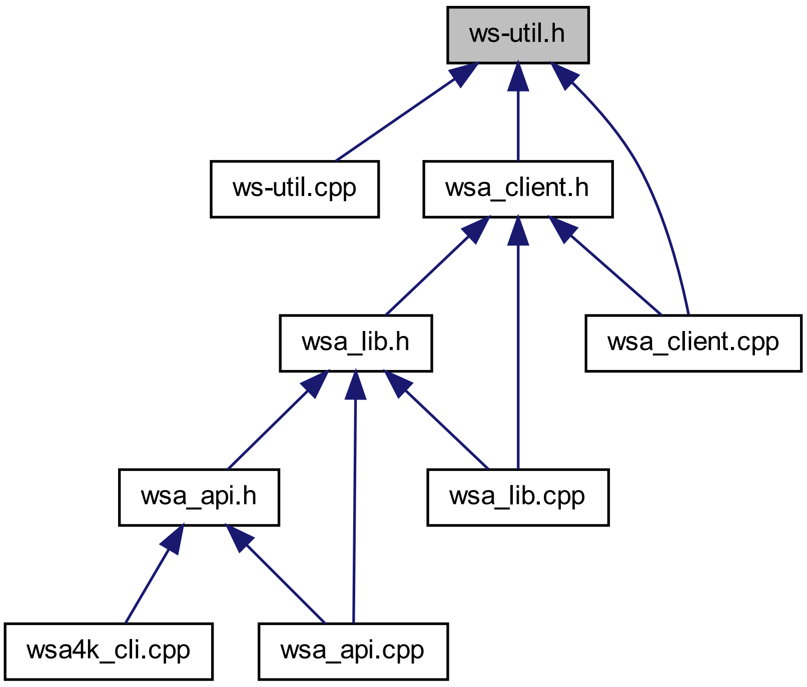
5.6.2.2 const int **kNumMessages** = sizeof(gaErrorList) / sizeof(ErrorEntry)

5.7 ws-util.h File Reference

Include dependency graph for ws-util.h:



This graph shows which files directly or indirectly include this file:



Functions

- const char ∗ WSAGetLastErrorMessage (const char ∗pcMessagePrefix, int nErrorID=0)
- bool ShutdownConnection (SOCKET sd, char ∗sock_name)

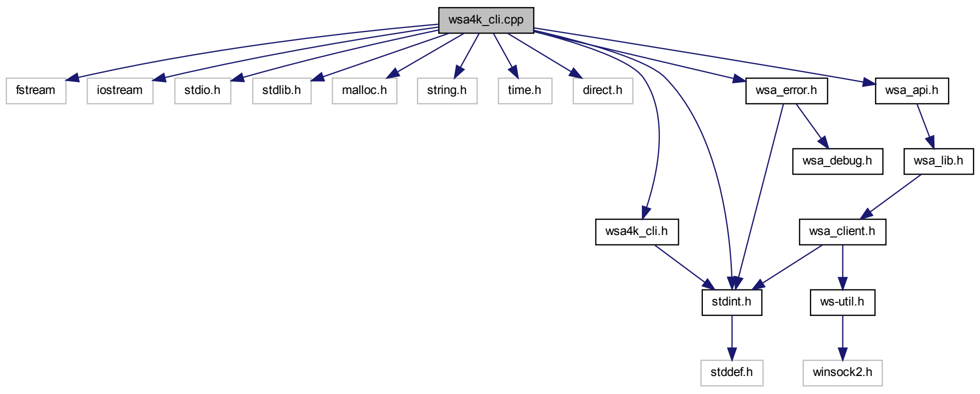
5.7.1 Function Documentation

5.7.1.1 bool ShutdownConnection (SOCKET *sd,* char ∗ *sock_name*)

5.7.1.2 const char∗ WSAGetLastErrorMessage (const char ∗ *pcMessagePrefix,* int *nErrorID =* 0)

5.8 wsa4k_cli.cpp File Reference

Include dependency graph for wsa4k_cli.cpp:



Functions

- int8_t process_cmds (struct wsa_device ∗dev, char ∗cmd_words[ ], int16_t num_-words)
- void print_cli_menu (struct wsa_device ∗dev)
- char ∗ get_input_cmd (uint8_t pretext)
- int8_t process_cmds (struct wsa_device ∗dev, char ∗∗cmd_words, int16_t num_-words)
- int16_t do_wsa (const char ∗wsa_addr)
- int16_t start_cli (void)
- int16_t process_call_mode (int32_t argc, char ∗∗argv)

5.8.1 Function Documentation

5.8.1.1 int16_t do_wsa (const char ∗ *wsa_addr*)

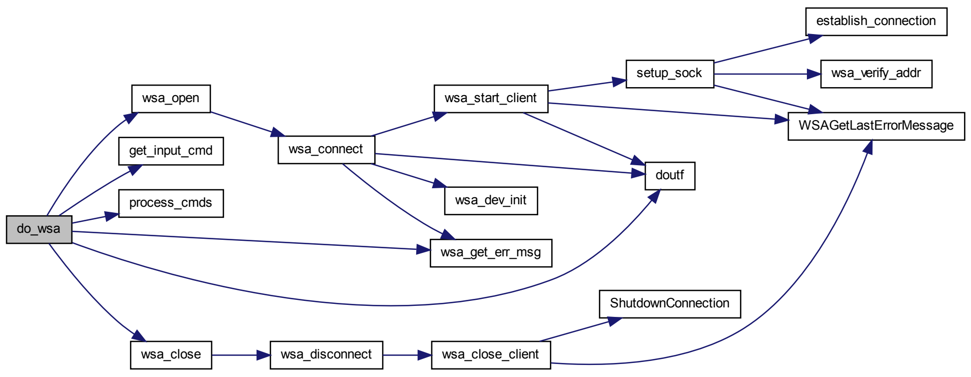
Setup WSA device variables, start the WSA connection and

Parameters

wsa_addr	- A char pointer to the IP address of the WSA

Returns

Here is the call graph for this function:



5.8.1.2 char ∗ get_input_cmd (uint8_t *pretext*)

Get input characters/string from the console and return the string all capitalized when the return key is pressed.

Parameters

pretext	- A TRUE or FALSE flag to indicate if the default "enter a command" text is to be printed.

Returns

The characters inputted.

5.8.1.3 void print_cli_menu (struct wsa_device ∗ *dev*)

Print out the CLI options menu

Parameters

dev	- a wsa device structure.

Returns

None

5.8.1.4 int16_t process_call_mode (int32_t *argc,* char ∗∗ *argv*)

Process the standalone call '-c' method Takes argument string in the form of: <executable name>=""> -c [-h] -ip=<...> [{h}] {...}

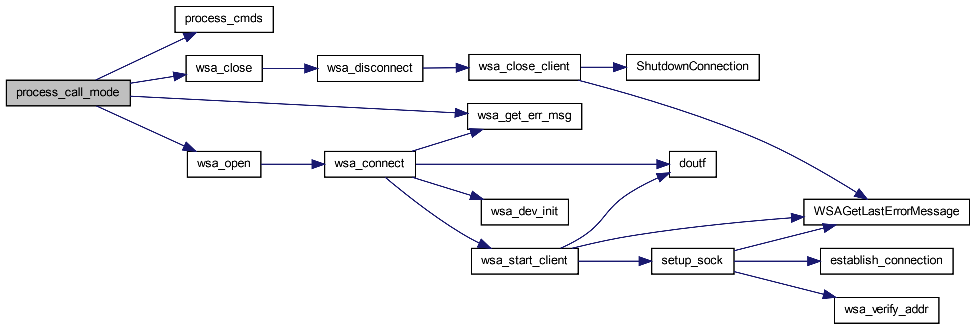
Parameters

argc	- Integer number of argument words
argv	- Pointer to pointer of characters

Returns

0 if success, negative number if failed

Here is the call graph for this function:



5.8.1.5 int8_t process_cmds (struct wsa_device * *dev*, char ** *cmd_words*, int16_t *num_words*)

Process any command (only) string.

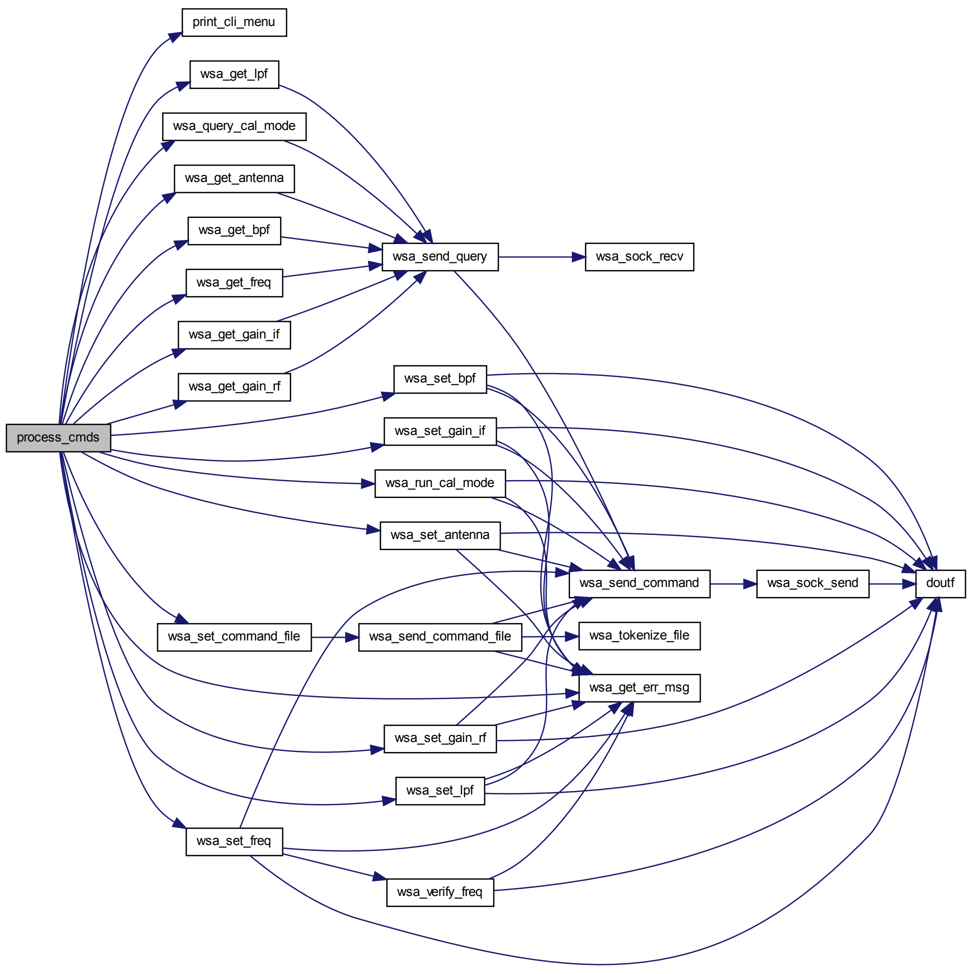
Parameters

dev	- A pointer to the WSA device structure.
cmd_words	- A pointer to pointers of char for storing command words.
num_words	- Number of words within the command.

Returns

1 if 'q'uit is set, 0 for no error.

Here is the call graph for this function:



5.8.1.6 int8_t process_cmds (struct wsa_device ∗ *dev,* char ∗ *cmd_words[ ],* int16_t *num_words*)

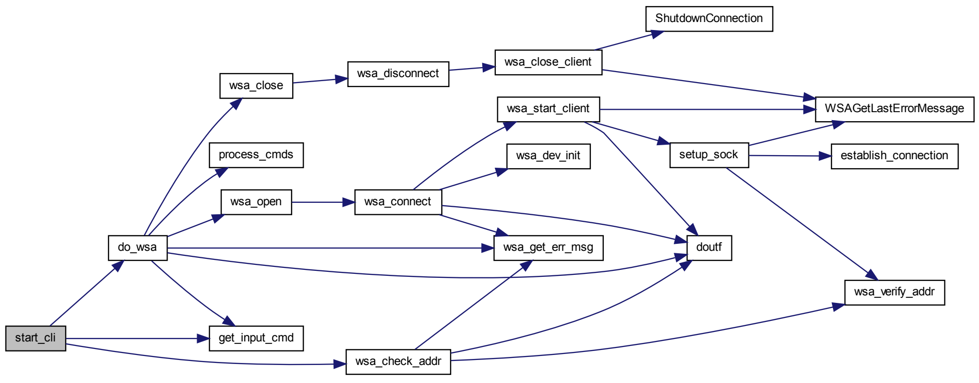
5.8.1.7 int16_t start_cli (void)

Start the CLI tool. First get a valid IP address from users, verify and start the WSA connection.

Returns

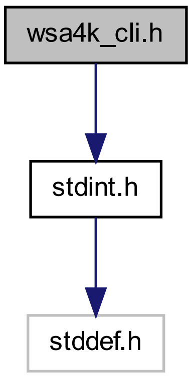
0 if successful

Here is the call graph for this function:

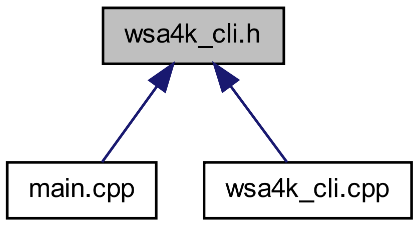


5.9 wsa4k_cli.h File Reference

Include dependency graph for wsa4k_cli.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define MAX_CMD_WORDS 5
- #define MAX_STR_LEN 200
- #define MAX_BUF_SIZE 20
- #define MAX_ANT_PORT 2
- #define MAX_FS 1000
- #define MHZ 1000000
- #define FALSE 0
- #define TRUE 1
- #define HISLIP 4880

Functions

- int16_t start_cli (void)
- int16_t process_call_mode (int32_t argc, char **argv)

Variables

- uint8_t debug_mode
- uint8_t call_mode

5.9.1 Define Documentation

5.9.1.1 #define FALSE 0

5.9.1.2 #define HISLIP 4880

5.9.1.3 #define MAX_ANT_PORT 2

5.9.1.4 #define MAX_BUF_SIZE 20

5.9.1.5 #define MAX_CMD_WORDS 5

5.9.1.6 #define MAX_FS 1000

5.9.1.7 #define MAX_STR_LEN 200

5.9.1.8 #define MHZ 1000000

5.9.1.9 #define TRUE 1

5.9.2 Function Documentation

5.9.2.1 int16_t process_call_mode (int32_t *argc*, char ∗∗ *argv*)

Process the standalone call '-c' method Takes argument string in the form of: <executable name>=""> -c [-h] -ip=<...> [{h}] {...}

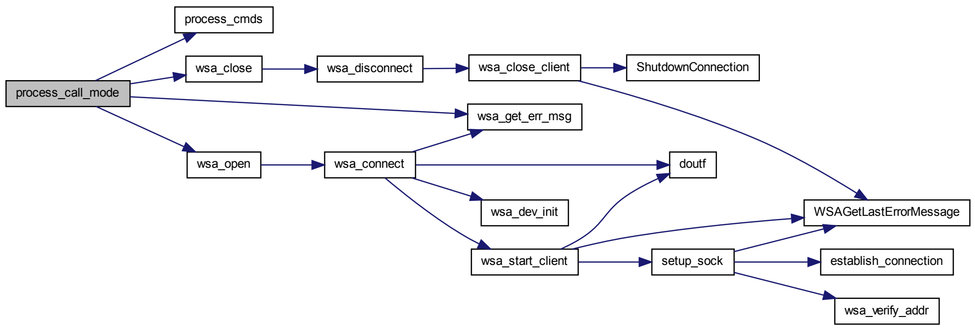
Parameters

argc	- Integer number of argument words
argv	- Pointer to pointer of characters

Returns

0 if success, negative number if failed

Here is the call graph for this function:



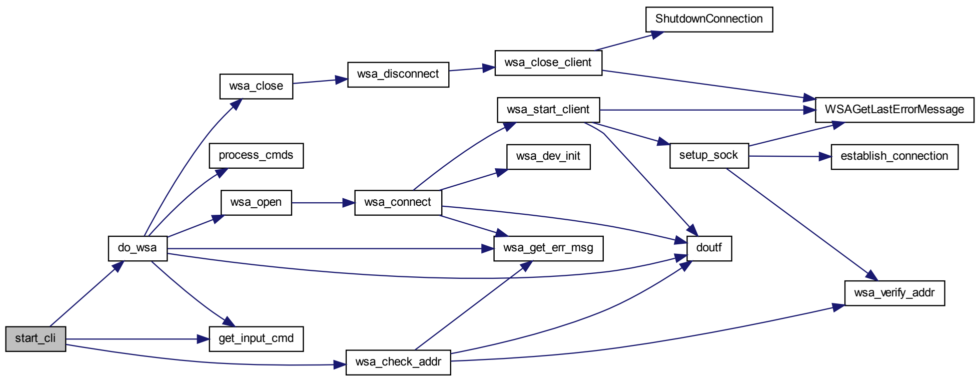
5.9.2.2 int16_t start_cli (void)

Start the CLI tool. First get a valid IP address from users, verify and start the WSA connection.

Returns

0 if successful

Here is the call graph for this function:



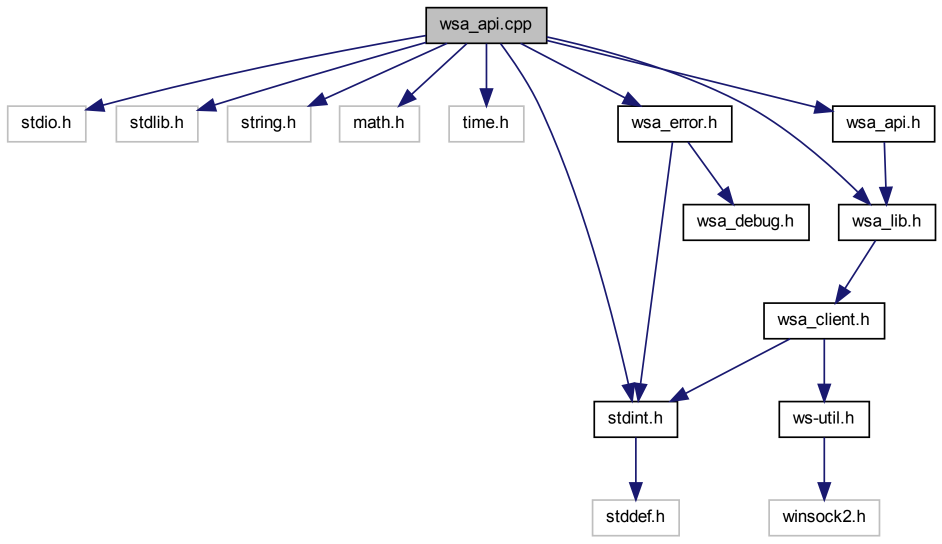
5.9.3 Variable Documentation

5.9.3.1 uint8_t call_mode

5.9.3.2 uint8_t debug_mode

5.10 wsa_api.cpp File Reference

Include dependency graph for wsa_api.cpp:



Defines

- #define MAX_ANT_PORT 2

Functions

- int16_t wsa_verify_freq (struct wsa_device ∗dev, uint64_t freq)
- int16_t wsa_open (struct wsa_device ∗dev, char ∗intf_method)
- void wsa_close (struct wsa_device ∗dev)
- int16_t wsa_check_addr (char ∗ip_addr)
- int16_t wsa_list (char ∗∗wsa_list)
- int16_t wsa_is_connected (struct wsa_device ∗dev)
- int16_t wsa_set_command_file (struct wsa_device ∗dev, char ∗file_name)
- float wsa_get_abs_max_amp (struct wsa_device ∗dev, wsa_gain gain)
- int64_t wsa_read_pkt (struct wsa_device ∗dev, struct wsa_frame_header ∗header, int16_t ∗i_buf, int16_t ∗q_buf, const uint64_t sample_size)
- int64_t wsa_get_freq (struct wsa_device ∗dev)
- int16_t wsa_set_freq (struct wsa_device ∗dev, uint64_t cfreq)
- float wsa_get_gain_if (struct wsa_device ∗dev)
- int16_t wsa_set_gain_if (struct wsa_device ∗dev, float gain)

- wsa_gain wsa_get_gain_rf (struct wsa_device ∗dev)
- int16_t wsa_set_gain_rf (struct wsa_device ∗dev, wsa_gain gain)
- int16_t wsa_get_antenna (struct wsa_device ∗dev)
- int16_t wsa_set_antenna (struct wsa_device ∗dev, uint8_t port_num)
- int16_t wsa_get_bpf (struct wsa_device ∗dev)
- int16_t wsa_set_bpf (struct wsa_device ∗dev, uint8_t mode)
- int16_t wsa_get_lpf (struct wsa_device ∗dev)
- int16_t wsa_set_lpf (struct wsa_device ∗dev, uint8_t mode)
- int16_t wsa_query_cal_mode (struct wsa_device ∗dev)
- int16_t wsa_run_cal_mode (struct wsa_device ∗dev, uint8_t mode)

5.10.1 Define Documentation

5.10.1.1 #define MAX_ANT_PORT 2

5.10.2 Function Documentation

5.10.2.1 int16_t wsa_check_addr (char ∗ *ip_addr*)

Verify if the IP address or host name given is valid for the WSA.

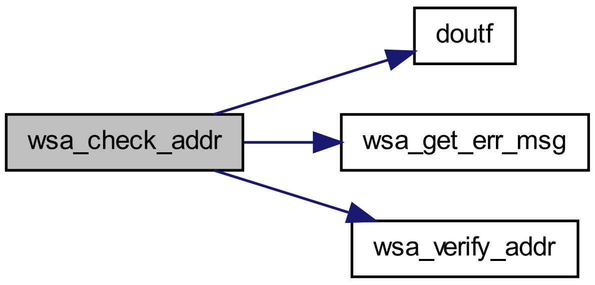
Parameters

ip_addr	- A char pointer to the IP address or host name to be verified.

Returns

1 if the IP is valid, or a negative number on error.

Here is the call graph for this function:



5.10.2.2 void wsa_close (struct **wsa_device** * *dev*)

Closes the device handle if one is opened and stops any existing data capture.

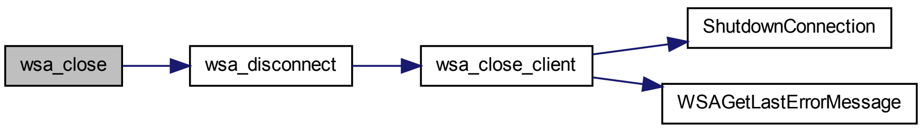
Parameters

dev	- A pointer to a WSA device structure to be closed.

Returns

none

Here is the call graph for this function:



5.10.2.3 float wsa_get_abs_max_amp (struct **wsa_device** * *dev*, **wsa_gain** *gain*)

Gets the absolute maximum RF input level (dBm) for the WSA at the given gain setting.

Operating the WSA device at the absolute maximum may cause damage to the device.

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain setting of **wsa_gain** type at which the absolute maximum amplitude input level is to be retrieved.

Returns

The absolute maximum RF input level in dBm or negative error number.

5.10.2.4 int16_t wsa_get_antenna (struct **wsa_device** * *dev*)

Gets which antenna port is currently in used with the RFE board.

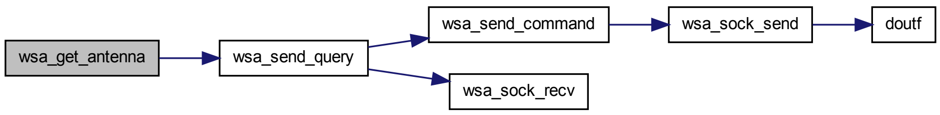
Parameters

dev	- A pointer to the WSA device structure.

Returns

The antenna port number on success, or a negative number on error.

Here is the call graph for this function:



5.10.2.5 int16_t wsa_get_bpf (struct wsa_device * *dev*)

Gets the current mode of the RFE's preselect BPF stage.

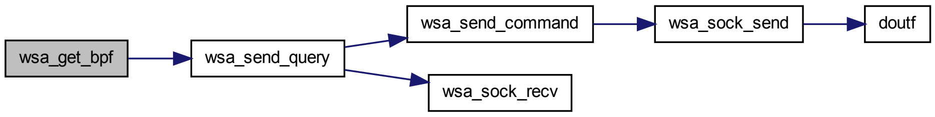
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 (on), 0 (off), or a negative number on error.

Here is the call graph for this function:



5.10.2.6 int64_t wsa_get_freq (struct wsa_device * *dev*)

Retrieves the center frequency that the WSA is running at.

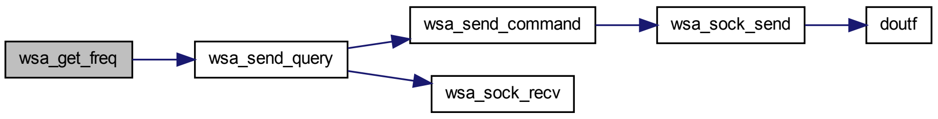
Parameters

dev	- A pointer to the WSA device structure.

Returns

The frequency in Hz, or a negative number on error.

Here is the call graph for this function:



5.10.2.7 float wsa_get_gain_if (struct wsa_device * *dev*)

Gets the current IF gain value of the RFE in dB.

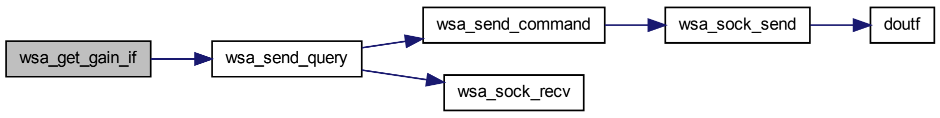
Parameters

dev	- A pointer to the WSA device structure.

Returns

The gain value in dB, or a large negative number on error.

Here is the call graph for this function:



5.10.2.8 wsa_gain wsa_get_gain_rf (struct wsa_device * *dev*)

Gets the current quantized RF front end gain setting of the RFE.

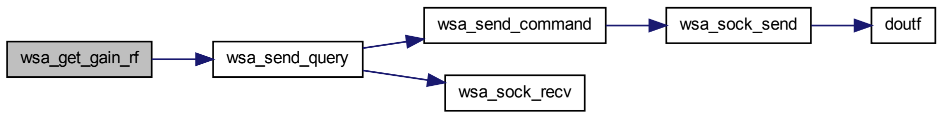
Parameters

dev	- A pointer to the WSA device structure.

Returns

The gain setting of wsa_gain type, or a negative number on error.

Here is the call graph for this function:



5.10.2.9 int16_t wsa_get_lpf (struct wsa_device * *dev*)

Gets the current mode of the RFE's internal anti-aliasing LPF.

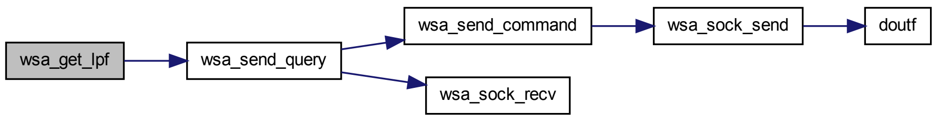
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 (on), 0 (off), or a negative number on error.

Here is the call graph for this function:



5.10.2.10 int16_t wsa_is_connected (struct wsa_device * *dev*)

Indicates if the WSA is still connected to the PC.

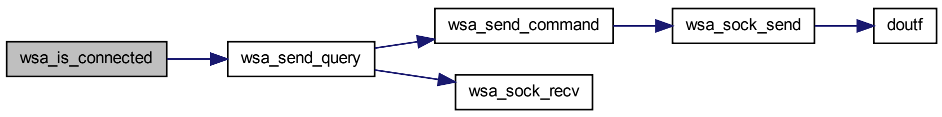
Parameters

dev	- A pointer to the WSA device structure to be verified for the connection.

Returns

1 if it is connected, 0 if not connected, or a negative number if errors.

Here is the call graph for this function:



5.10.2.11 int16_t wsa_list (char ** *wsa_list*)

Count and print out the IPs of connected WSAs to the network? or the PC??? For now, will list the IPs for any of the connected devices to a PC?

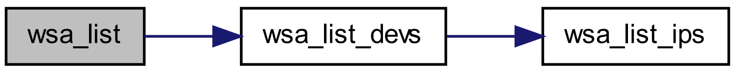
Parameters

wsa_list	- A double char pointer to store (WSA???) IP addresses connected to a network???.

Returns

Number of connected WSAs (or IPs for now) on success, or a negative number on error.

Here is the call graph for this function:



5.10.2.12 int16_t wsa_open (struct wsa_device * *dev,* char * *intf_method*)

Establishes a connection of choice specified by the interface method to the WSA.

At success, the handle remains open for future access by other library methods until wsa_close() is called. When unsuccessful, the WSA will be closed automatically and an error is returned.

Parameters

dev	- A pointer to the WSA device structure to be opened.

intf_method	- A char pointer to store the interface method to the WSA. Possible methods: • With LAN, use: "TCPIP::<Ip address of the WSA>::HISLIP" • With USB, use: "USB" (check if supported with the WSA version used).

Returns

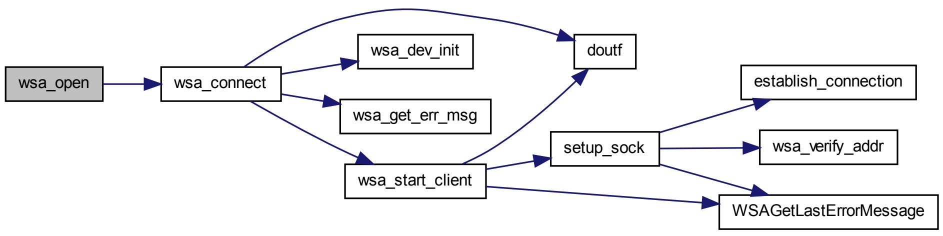
0 on success, or a negative number on error.

Errors:

Situations that will generate an error are:

- the interface method does not exist for the WSA product version.
- the WSA is not detected (has not been connected or powered up).
-

Here is the call graph for this function:



5.10.2.13 int16_t wsa_query_cal_mode (struct wsa_device * *dev*)

Checks if the RFE's internal calibration has finished or not.

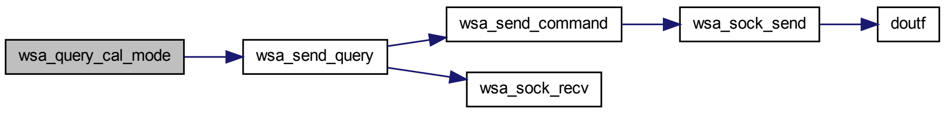
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 if the calibration is still running or 0 if completed, or a negative number on error.

Here is the call graph for this function:



5.10.2.14 int64_t wsa_read_pkt (struct **wsa_device** * *dev*, struct **wsa_frame_header** * *header*, int16_t * *i_buf*, int16_t * *q_buf*, const uint64_t *sample_size*)

Reads a frame of data. *Each* frame consists of a header, and I and Q buffers of data of length determine by the **sample_size** parameter.

Parameters

dev	- A pointer to the WSA device structure.
header	- A pointer to **wsa_frame_header** structure to store information for the frame.
i_buf	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the sample_size.
q_buf	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the sample_size.
sample_size	- A 64-bit unsigned integer sample size (i.e. {I, Q} sample pairs) per data frame to be captured. The frame size is limited to a maximum number, **max_sample_size**, listed in the **wsa_descriptor** structure.

Returns

The number of data samples read upon success, or a negative number on error.

5.10.2.15 int16_t wsa_run_cal_mode (struct **wsa_device** * *dev*, uint8_t *mode*)

Runs the RFE'S internal calibration mode or cancel it.

While the calibration mode is running, no other commands should be running until the calibration is finished by using wsa_query_cal_mode(), or could be cancelled

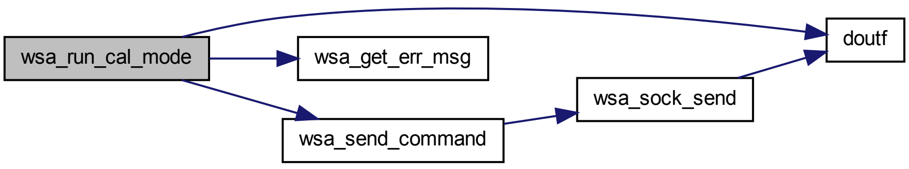
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 1 - Run, 0 - Cancel.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.10.2.16 int16_t wsa_set_antenna (struct wsa_device * *dev*, uint8_t *port_num*)

Sets the antenna port to be used for the RFE board.

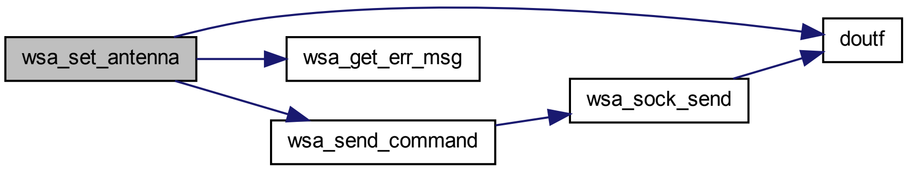
Parameters

dev	- A pointer to the WSA device structure.
port_num	- An integer port number to used. Available ports: 1, 2. Or see product datasheet for ports availability. **Note:** When calibration mode is enabled through wsa_run_cal_mode(), these antenna ports will not be available. The seletected port will resume when the calibration mode is set to off.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.10.2.17 int16_t wsa_set_bpf (struct wsa_device * *dev*, uint8_t *mode*)

Sets the RFE's preselect band pass filter (BPF) stage on or off (bypassing).

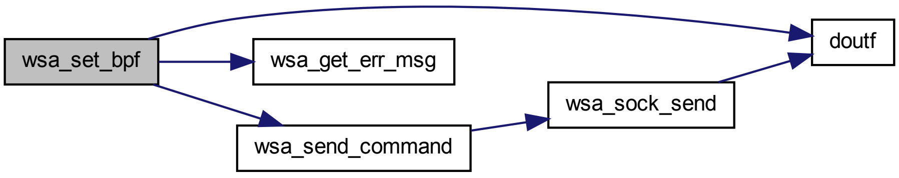
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 0 - Off, 1 - On.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.10.2.18 int16_t wsa_set_command_file (struct wsa_device ∗ *dev*, char ∗ *file_name*)

Read command line(s) stored in the given **file_name** and set each line to the WSA.

Remarks

- Assuming each command line is for a single function followed by a new line.
- Currently read only SCPI commands. Other types of commands, TBD.

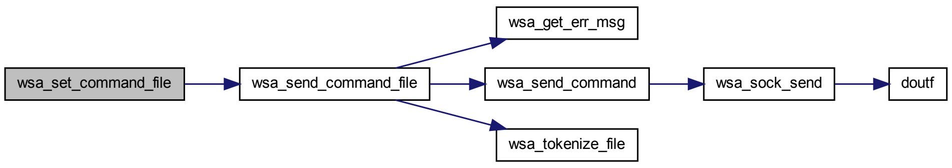
Parameters

dev	- A pointer to the WSA device structure.
file_name	- A pointer to the file name

Returns

Number of command lines at success, or a negative error number.

Here is the call graph for this function:



5.10.2.19 int16_t wsa_set_freq (struct wsa_device * *dev,* uint64_t *cfreq*)

Sets the WSA to the desired center frequency, **cfreq**.

Remarks

> **wsa_set_freq()** will return error if trigger mode is already running. See the **descr** component of **wsa_dev** structure for maximum/minimum frequency values.

Parameters

dev	- A pointer to the WSA device structure.
cfreq	- The center frequency to set, in Hz

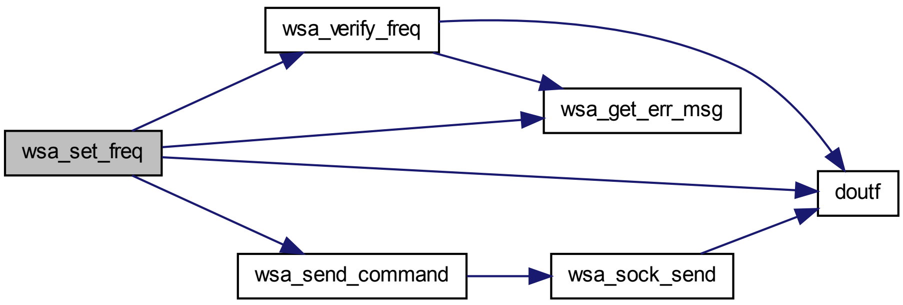
Returns

> 0 on success, or a negative number on error.

Errors:

- Frequency out of range.
- Set frequency when WSA is in trigger mode.
- Incorrect frequency resolution (check with data sheet).

Here is the call graph for this function:



5.10.2.20 int16_t wsa_set_gain_if (struct wsa_device ∗ *dev,* float *gain*)

Sets the gain value in dB for the variable IF gain stages of the RFE, which is additive to the primary RF quantized gain stages (wsa_set_gain_rf()).

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain level in dB.

Remarks

See the **descr** component of **wsa_dev** structure for maximum/minimum IF gain values. ???

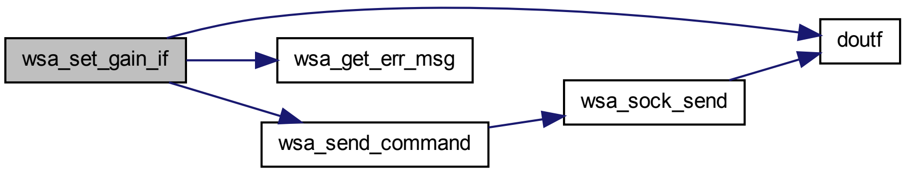
Returns

0 on success, or a negative number on error.

Errors:

- Gain level out of range.

Here is the call graph for this function:



5.10.2.21 int16_t wsa_set_gain_rf (struct wsa_device * *dev*, wsa_gain *gain*)

Sets the quantized **gain** (sensitivity) level for the RFE of the WSA.

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain setting of type wsa_gain to set for WSA. Valid gain settings are: • WSA_GAIN_HIGH • WSA_GAIN_MEDIUM • WSA_GAIN_LOW • WSA_GAIN_VLOW

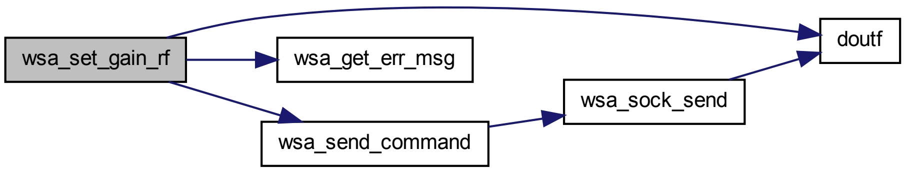
Returns

0 on success, or a negative number on error.

Errors:

- Gain setting not allow.

Here is the call graph for this function:



5.10.2.22 int16_t wsa_set_lpf (struct wsa_device * *dev*, uint8_t *mode*)

Sets the internal anti-aliasing low pass filter (LPF) on or off (bypassing).

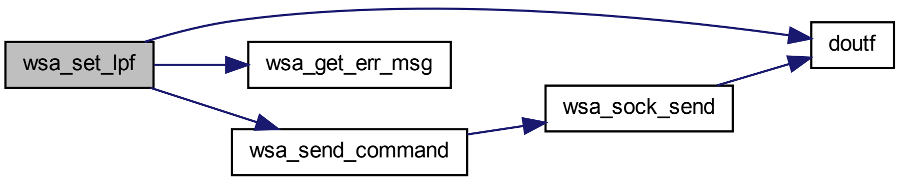
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 0 - Off, 1 - On.

Returns

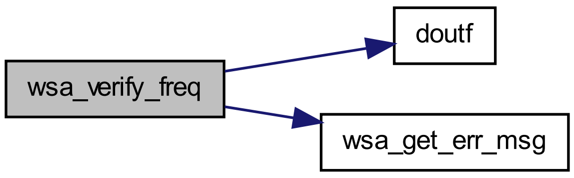
0 on success, or a negative number on error.

Here is the call graph for this function:



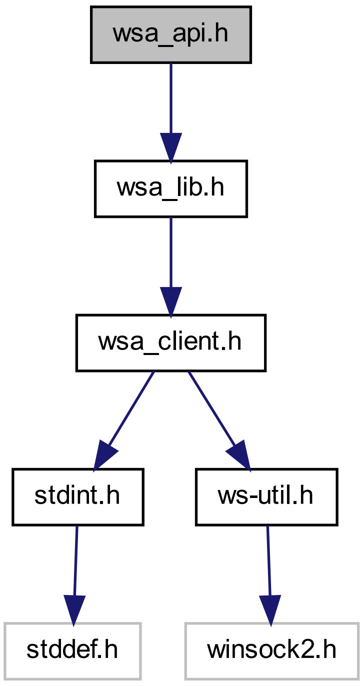
5.10.2.23 int16_t wsa_verify_freq (struct wsa_device * *dev*, uint64_t *freq*)

Here is the call graph for this function:

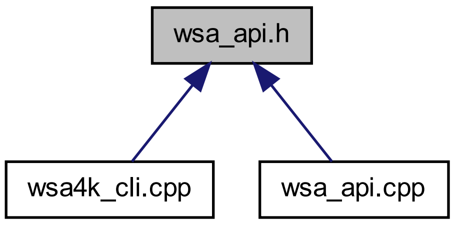


5.11 wsa_api.h File Reference

Include dependency graph for wsa_api.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct wsa_descriptor

  This structure stores WSA information.
- struct wsa_time

  This structure contains the time information. It is used for the time stamp in a frame header.
- struct wsa_frame_header

  This structure contains header information related to each frame read by wsa_get_-frame().
- struct wsa_socket

  A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition.
- struct wsa_device

  A structure containing the components associate with each WSA device.

Enumerations

- enum wsa_gain {

  WSA_GAIN_HIGH = 1, WSA_GAIN_MEDIUM, WSA_GAIN_LOW, WSA_GAIN_-VLOW,

  WSA_GAIN_HIGH = 1, WSA_GAIN_MEDIUM, WSA_GAIN_LOW, WSA_GAIN_-VLOW }

Functions

- int16_t wsa_open (struct wsa_device ∗dev, char ∗intf_method)
- void wsa_close (struct wsa_device ∗dev)

- int16_t wsa_check_addr (char ∗intf_method)
- int16_t wsa_list (char ∗∗wsa_list)
- int16_t wsa_is_connected (struct wsa_device ∗dev)
- int16_t wsa_set_command_file (struct wsa_device ∗dev, char ∗file_name)
- float wsa_get_abs_max_amp (struct wsa_device ∗dev, wsa_gain gain)
- int64_t wsa_read_pkt (struct wsa_device ∗dev, struct wsa_frame_header ∗header, int16_t ∗i_buf, int16_t ∗q_buf, const uint64_t sample_size)
- int64_t wsa_get_freq (struct wsa_device ∗dev)
- int16_t wsa_set_freq (struct wsa_device ∗dev, uint64_t cfreq)
- float wsa_get_gain_if (struct wsa_device ∗dev)
- int16_t wsa_set_gain_if (struct wsa_device ∗dev, float gain)
- wsa_gain wsa_get_gain_rf (struct wsa_device ∗dev)
- int16_t wsa_set_gain_rf (struct wsa_device ∗dev, wsa_gain gain)
- int16_t wsa_get_antenna (struct wsa_device ∗dev)
- int16_t wsa_set_antenna (struct wsa_device ∗dev, uint8_t port_num)
- int16_t wsa_get_bpf (struct wsa_device ∗dev)
- int16_t wsa_set_bpf (struct wsa_device ∗dev, uint8_t mode)
- int16_t wsa_get_lpf (struct wsa_device ∗dev)
- int16_t wsa_set_lpf (struct wsa_device ∗dev, uint8_t mode)
- int16_t wsa_query_cal_mode (struct wsa_device ∗dev)
- int16_t wsa_run_cal_mode (struct wsa_device ∗dev, uint8_t mode)

5.11.1 Enumeration Type Documentation

5.11.1.1 enum wsa_gain

Defines the RF quantized gain settings available for the radio front end (RFE) of the WSA.

Enumerator:

WSA_GAIN_HIGH	High RF amplification. Value 1.
WSA_GAIN_MEDIUM	Medium RF amplification.
WSA_GAIN_LOW	Low RF amplification.
WSA_GAIN_VLOW	Very low RF amplification.
WSA_GAIN_HIGH	
WSA_GAIN_MEDIUM	
WSA_GAIN_LOW	
WSA_GAIN_VLOW	

5.11.2 Function Documentation

5.11.2.1 int16_t wsa_check_addr (char ∗ *ip_addr*)

Verify if the IP address or host name given is valid for the WSA.

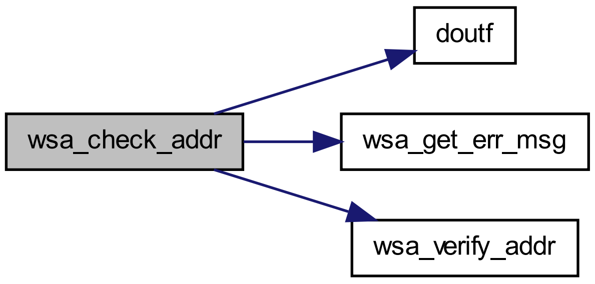
Parameters

ip_addr	- A char pointer to the IP address or host name to be verified.

Returns

1 if the IP is valid, or a negative number on error.

Here is the call graph for this function:



5.11.2.2 void wsa_close (struct wsa_device * *dev*)

Closes the device handle if one is opened and stops any existing data capture.

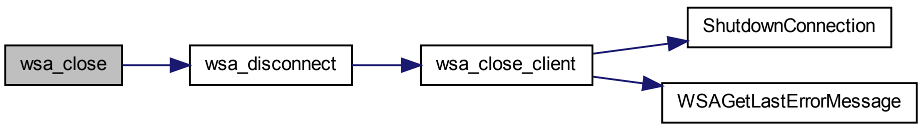
Parameters

dev	- A pointer to a WSA device structure to be closed.

Returns

none

Here is the call graph for this function:



5.11.2.3 float wsa_get_abs_max_amp (struct wsa_device * *dev*, wsa_gain *gain*)

Gets the absolute maximum RF input level (dBm) for the WSA at the given gain setting.

Operating the WSA device at the absolute maximum may cause damage to the device.

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain setting of **wsa_gain** type at which the absolute maximum amplitude input level is to be retrieved.

Returns

The absolute maximum RF input level in dBm or negative error number.

5.11.2.4 int16_t wsa_get_antenna (struct wsa_device * *dev*)

Gets which antenna port is currently in used with the RFE board.

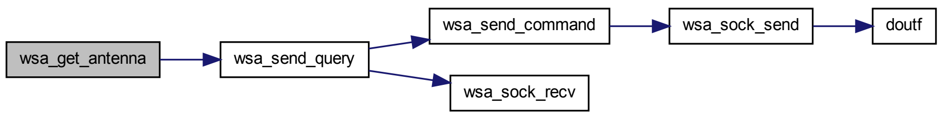
Parameters

dev	- A pointer to the WSA device structure.

Returns

The antenna port number on success, or a negative number on error.

Here is the call graph for this function:



5.11.2.5 int16_t wsa_get_bpf (struct wsa_device * *dev*)

Gets the current mode of the RFE's preselect BPF stage.

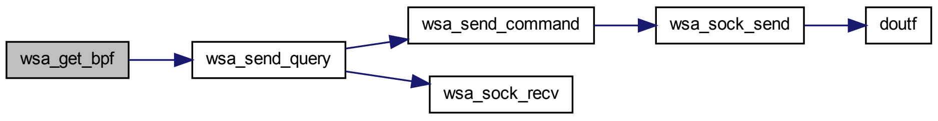
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 (on), 0 (off), or a negative number on error.

Here is the call graph for this function:



5.11.2.6 int64_t wsa_get_freq (struct wsa_device * *dev*)

Retrieves the center frequency that the WSA is running at.

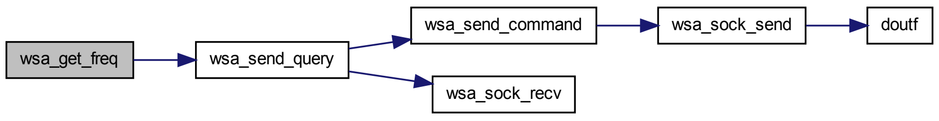
Parameters

dev	- A pointer to the WSA device structure.

Returns

The frequency in Hz, or a negative number on error.

Here is the call graph for this function:



5.11.2.7 float wsa_get_gain_if (struct wsa_device * *dev*)

Gets the current IF gain value of the RFE in dB.

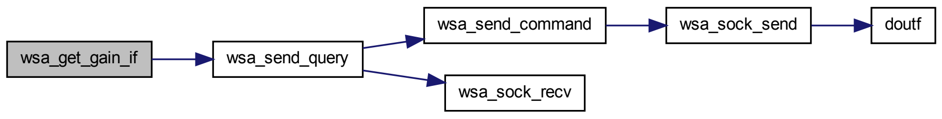
Parameters

dev	- A pointer to the WSA device structure.

Returns

The gain value in dB, or a large negative number on error.

Here is the call graph for this function:



5.11.2.8 wsa_gain wsa_get_gain_rf (struct wsa_device * *dev*)

Gets the current quantized RF front end gain setting of the RFE.

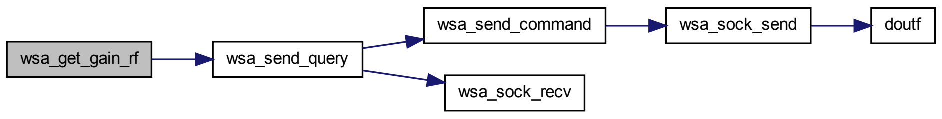
Parameters

dev	- A pointer to the WSA device structure.

Returns

The gain setting of wsa_gain type, or a negative number on error.

Here is the call graph for this function:



5.11.2.9 int16_t wsa_get_lpf (struct wsa_device * *dev*)

Gets the current mode of the RFE's internal anti-aliasing LPF.

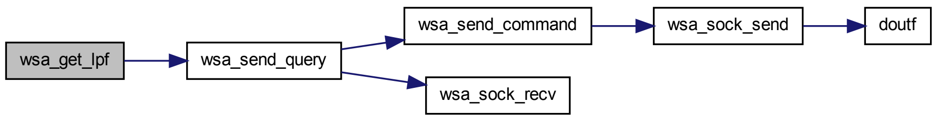
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 (on), 0 (off), or a negative number on error.

Here is the call graph for this function:



5.11.2.10 int16_t wsa_is_connected (struct wsa_device * *dev*)

Indicates if the WSA is still connected to the PC.

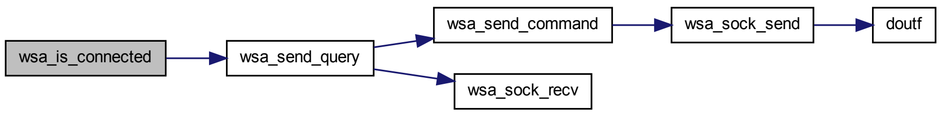
Parameters

dev	- A pointer to the WSA device structure to be verified for the connection.

Returns

1 if it is connected, 0 if not connected, or a negative number if errors.

Here is the call graph for this function:



5.11.2.11 int16_t wsa_list (char ** *wsa_list*)

Count and print out the IPs of connected WSAs to the network? or the PC??? For now, will list the IPs for any of the connected devices to a PC?

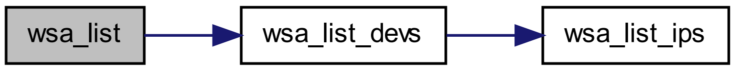
Parameters

wsa_list	- A double char pointer to store (WSA???) IP addresses connected to a network???.

Returns

Number of connected WSAs (or IPs for now) on success, or a negative number on error.

Here is the call graph for this function:



5.11.2.12 int16_t wsa_open (struct wsa_device * *dev*, char * *intf_method*)

Establishes a connection of choice specified by the interface method to the WSA.

At success, the handle remains open for future access by other library methods until wsa_close() is called. When unsuccessful, the WSA will be closed automatically and an error is returned.

Parameters

dev	- A pointer to the WSA device structure to be opened.
intf_method	- A char pointer to store the interface method to the WSA. Possible methods: • With LAN, use: "TCPIP::<Ip address of the WSA>::HISLIP" • With USB, use: "USB" (check if supported with the WSA version used).

Returns

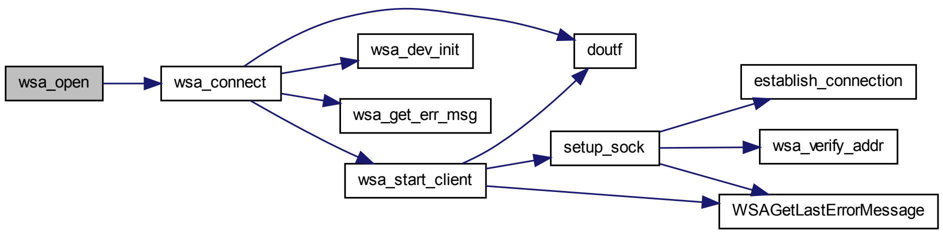
0 on success, or a negative number on error.

Errors:

Situations that will generate an error are:

- the interface method does not exist for the WSA product version.
- the WSA is not detected (has not been connected or powered up).
-

Here is the call graph for this function:



5.11.2.13 int16_t wsa_query_cal_mode (struct wsa_device * *dev*)

Checks if the RFE's internal calibration has finished or not.

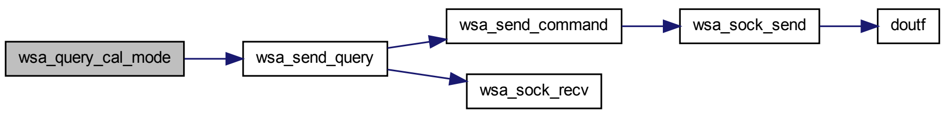
Parameters

dev	- A pointer to the WSA device structure.

Returns

1 if the calibration is still running or 0 if completed, or a negative number on error.

Here is the call graph for this function:



5.11.2.14 int64_t wsa_read_pkt (struct wsa_device * *dev*, struct wsa_frame_header * *header*, int16_t * *i_buf*, int16_t * *q_buf*, const uint64_t *sample_size*)

Reads a frame of data. *Each* frame consists of a header, and I and Q buffers of data of length determine by the **sample_size** parameter.

Parameters

dev	- A pointer to the WSA device structure.
header	- A pointer to **wsa_frame_header** structure to store information for the frame.

i_buf	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the sample_size.
q_buf	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the sample_size.
sample_size	- A 64-bit unsigned integer sample size (i.e. {I, Q} sample pairs) per data frame to be captured. The frame size is limited to a maximum number, **max_sample_size**, listed in the **wsa_descriptor** structure.

Returns

The number of data samples read upon success, or a negative number on error.

5.11.2.15 int16_t wsa_run_cal_mode (struct wsa_device * *dev,* uint8_t *mode*)

Runs the RFE'S internal calibration mode or cancel it.

While the calibration mode is running, no other commands should be running until the calibration is finished by using wsa_query_cal_mode(), or could be cancelled

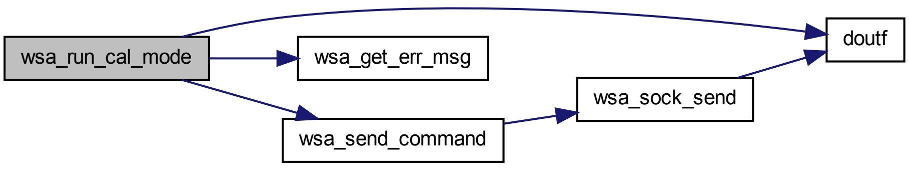
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 1 - Run, 0 - Cancel.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.11.2.16 int16_t wsa_set_antenna (struct wsa_device * *dev,* uint8_t *port_num*)

Sets the antenna port to be used for the RFE board.

Parameters

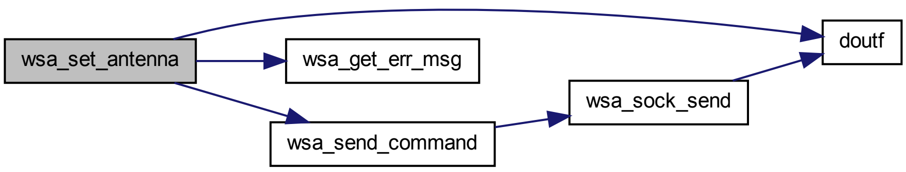
dev	- A pointer to the WSA device structure.

port_num	- An integer port number to used. Available ports: 1, 2. Or see product datasheet for ports availability. **Note:** When calibration mode is enabled through wsa_run_cal_mode(), these antenna ports will not be available. The seletected port will resume when the calibration mode is set to off.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.11.2.17 int16_t wsa_set_bpf (struct wsa_device * *dev,* uint8_t *mode*)

Sets the RFE's preselect band pass filter (BPF) stage on or off (bypassing).

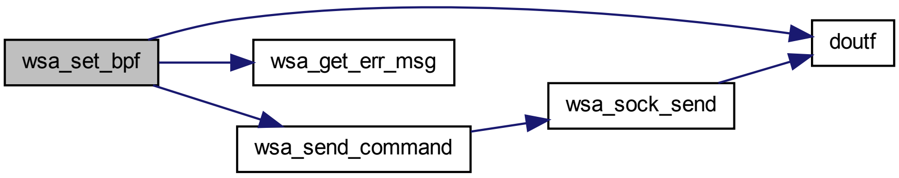
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 0 - Off, 1 - On.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.11.2.18 int16_t wsa_set_command_file (struct wsa_device * *dev*, char * *file_name*)

Read command line(s) stored in the given **file_name** and set each line to the WSA.

Remarks

- Assuming each command line is for a single function followed by a new line.
- Currently read only SCPI commands. Other types of commands, TBD.

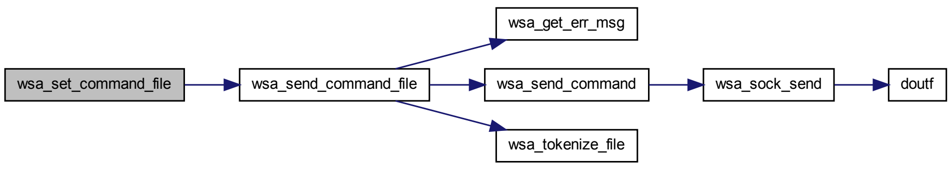
Parameters

dev	- A pointer to the WSA device structure.
file_name	- A pointer to the file name

Returns

Number of command lines at success, or a negative error number.

Here is the call graph for this function:



5.11.2.19 int16_t wsa_set_freq (struct wsa_device * *dev*, uint64_t *cfreq*)

Sets the WSA to the desired center frequency, **cfreq**.

Remarks

wsa_set_freq() will return error if trigger mode is already running. See the **descr** component of **wsa_dev** structure for maximum/minimum frequency values.

Parameters

dev	- A pointer to the WSA device structure.
cfreq	- The center frequency to set, in Hz

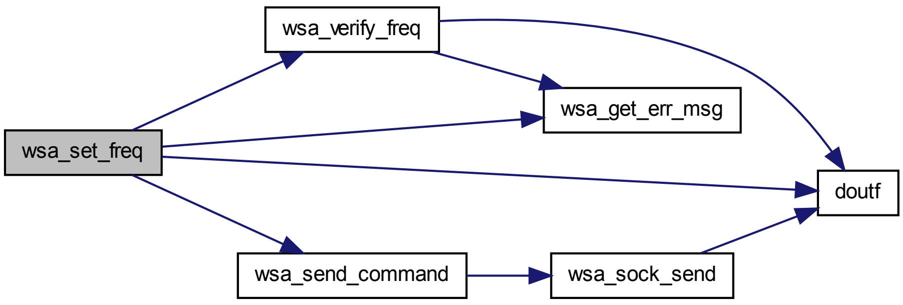
Returns

0 on success, or a negative number on error.

Errors:

- Frequency out of range.
- Set frequency when WSA is in trigger mode.
- Incorrect frequency resolution (check with data sheet).

Here is the call graph for this function:



5.11.2.20 int16_t wsa_set_gain_if (struct wsa_device ∗ *dev,* float *gain*)

Sets the gain value in dB for the variable IF gain stages of the RFE, which is additive to the primary RF quantized gain stages (wsa_set_gain_rf()).

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain level in dB.

Remarks

See the **descr** component of **wsa_dev** structure for maximum/minimum IF gain values. ???

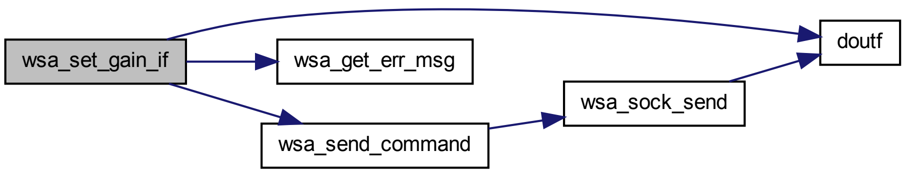
Returns

0 on success, or a negative number on error.

Errors:

- Gain level out of range.

Here is the call graph for this function:



5.11.2.21 int16_t wsa_set_gain_rf (struct wsa_device * *dev*, wsa_gain *gain*)

Sets the quantized **gain** (sensitivity) level for the RFE of the WSA.

Parameters

dev	- A pointer to the WSA device structure.
gain	- The gain setting of type wsa_gain to set for WSA. Valid gain settings are: • WSA_GAIN_HIGH • WSA_GAIN_MEDIUM • WSA_GAIN_LOW • WSA_GAIN_VLOW

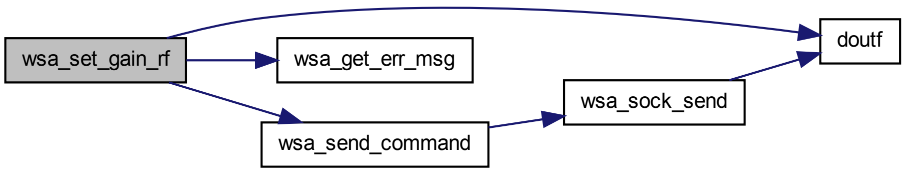
Returns

0 on success, or a negative number on error.

Errors:

- Gain setting not allow.

Here is the call graph for this function:



5.11.2.22 int16_t wsa_set_lpf (struct wsa_device ∗ *dev*, uint8_t *mode*)

Sets the internal anti-aliasing low pass filter (LPF) on or off (bypassing).

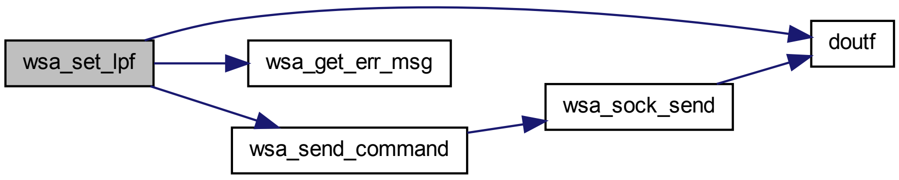
Parameters

dev	- A pointer to the WSA device structure.
mode	- An integer mode of selection: 0 - Off, 1 - On.

Returns

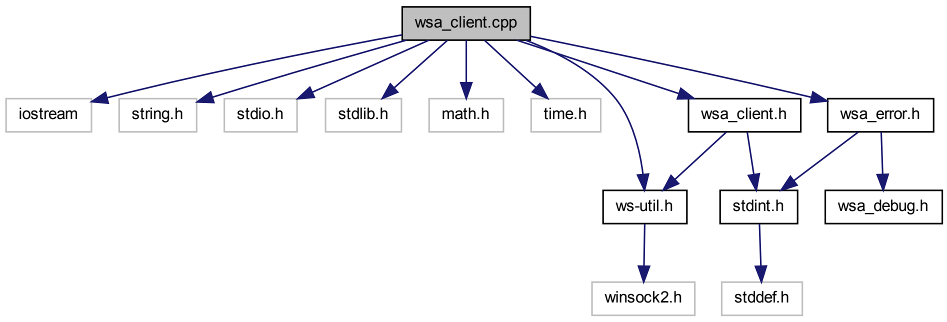
0 on success, or a negative number on error.

Here is the call graph for this function:



5.12 wsa_client.cpp File Reference

Include dependency graph for wsa_client.cpp:



Defines

- #define SHUTDOWN_DELAY

Functions

- SOCKET setup_sock (char ∗sock_name, const char ∗sock_addr, int32_t sock_port)
- SOCKET establish_connection (u_long sock_addr, u_short sock_port)
- int16_t wsa_start_client (const char ∗wsa_addr, SOCKET ∗cmd_sock, SOCKET ∗data_sock)
- int16_t wsa_close_client (SOCKET cmd_sock, SOCKET data_sock)
- u_long wsa_verify_addr (const char ∗sock_addr)
- int16_t wsa_sock_send (SOCKET out_sock, char ∗out_str, int32_t len)
- int64_t wsa_sock_recv (SOCKET in_sock, char ∗rx_buf_ptr, uint32_t time_out)
- int16_t wsa_sock_recv_words (SOCKET in_sock, char ∗rx_buf_ptr[ ], uint32_t time_out)
- uint8_t get_sock_ack (SOCKET in_sock, char ∗ack_str, long time_out)
- int16_t wsa_get_host_info (char ∗name)
- int16_t wsa_list_ips (char ∗∗ip_list)

Variables

- const int8_t kShutdownDelay = 1
- uint8_t debug_mode = FALSE
- uint8_t call_mode = FALSE
- char ∗ start = "STARTDATA\0"

- char ∗ stop = "STOPDATA\0"
- const int32_t cmd_port = 7000
- const int32_t data_port = 7000

5.12.1 Define Documentation

5.12.1.1 #define SHUTDOWN_DELAY

5.12.2 Function Documentation

5.12.2.1 SOCKET establish_connection (u_long *sock_addr*, u_short *sock_port*)

Connects to a given address, on a given port, both of which must be in network byte order. Returns

Parameters

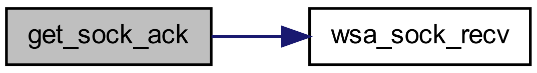
sock_addr	-
sock_port	-

Returns

Newly-connected socket when succeed, or INVALID_SOCKET when fail.

5.12.2.2 uint8_t get_sock_ack (SOCKET *in_sock*, char ∗ *ack_str*, long *time_out*)

Here is the call graph for this function:



5.12.2.3 SOCKET setup_sock (char ∗ *sock_name*, const char ∗ *sock_addr*, int32_t *sock_port*)

Look up, verify and establish the socket once deemed valid

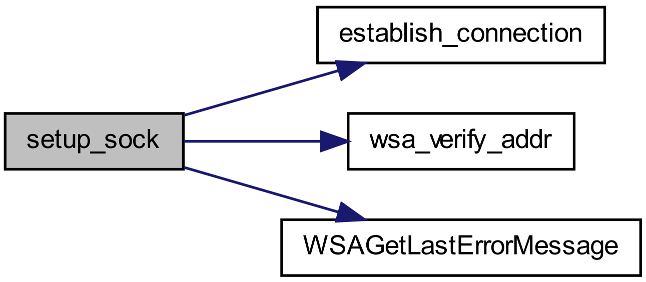
Parameters

sock_name	- Name of the socket (ex. server, client)
sock_addr	-
sock_port	-

Returns

Newly-connected socket when succeed, or INVALID_SOCKET when fail.

Here is the call graph for this function:



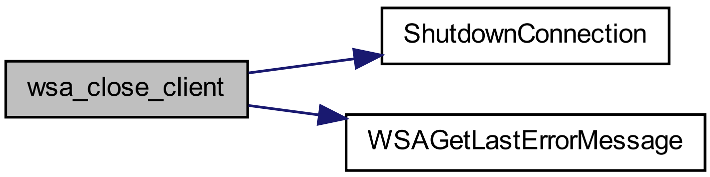
5.12.2.4 int16_t wsa_close_client (SOCKET *cmd_sock*, SOCKET *data_sock*)

Parameters

cmd_sock	-
data_sock	-

Returns

Here is the call graph for this function:



5.12.2.5 int16_t wsa_get_host_info (char * *name*)

Get host information based on the name given either as IP or host name format

Parameters

name	-

Returns

5.12.2.6 int16_t wsa_list_ips (char ** *ip_list*)

Print a list of host names and the associated IP available to a user's PC.

Parameters

ip_list	-

Returns

Number of IP addresses available.

5.12.2.7 int64_t wsa_sock_recv (SOCKET *in_sock,* char * *rx_buf_ptr,* uint32_t *time_out*)

Gets incoming strings from the server socket ? bytes at a time

Parameters

in_sock	-
rx_buf_ptr	-
time_out	- Time out in milliseconds

Returns

Number of "words" read

5.12.2.8 int16_t wsa_sock_recv_words (SOCKET *in_sock,* char * *rx_buf_ptr[ ],* uint32_t *time_out*)

5.12.2.9 int16_t wsa_sock_send (SOCKET *out_sock,* char * *out_str,* int32_t *len*)

Sends a string to the server.

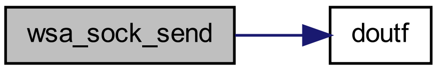
Parameters

out_sock	-
out_str	-
len	-

Returns

Number of bytes sent on success, or negative otherwise.

Here is the call graph for this function:



5.12.2.10 int16_t wsa_start_client (const char ∗ *wsa_addr,* SOCKET ∗ *cmd_sock,* SOCKET ∗ *data_sock*)

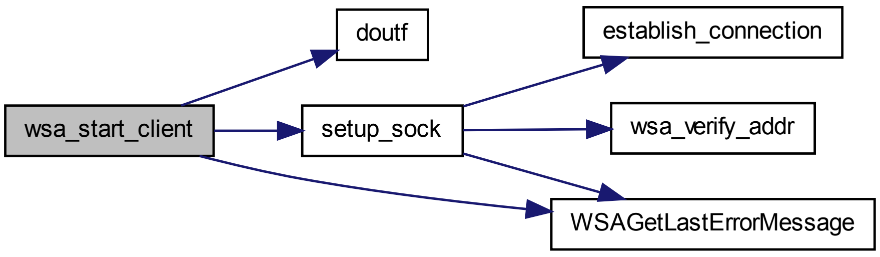
Call functions to initialize the sockets

Parameters

wsa_addr	-
cmd_sock	-
data_sock	-

Returns

Here is the call graph for this function:



5.12.2.11 u_long wsa_verify_addr (const char ∗ *sock_addr*)

Given an address string, determine if it's a dotted-quad IP address or a domain address. If the latter, ask DNS to resolve it. In either case, return resolved IP address. If we fail, we return INADDR_NONE.

Parameters

sock_addr	-

Returns

Resolved IP address or INADDR_NONE when failed.

5.12.3 Variable Documentation

5.12.3.1 uint8_t call_mode = FALSE

5.12.3.2 const int32_t cmd_port = 7000

5.12.3.3 const int32_t data_port = 7000

5.12.3.4 uint8_t debug_mode = FALSE

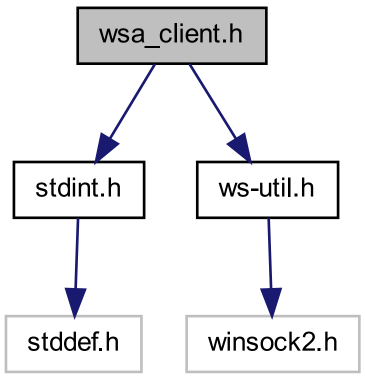
5.12.3.5 const int8_t kShutdownDelay = 1

5.12.3.6 char∗ start = "STARTDATA\0"

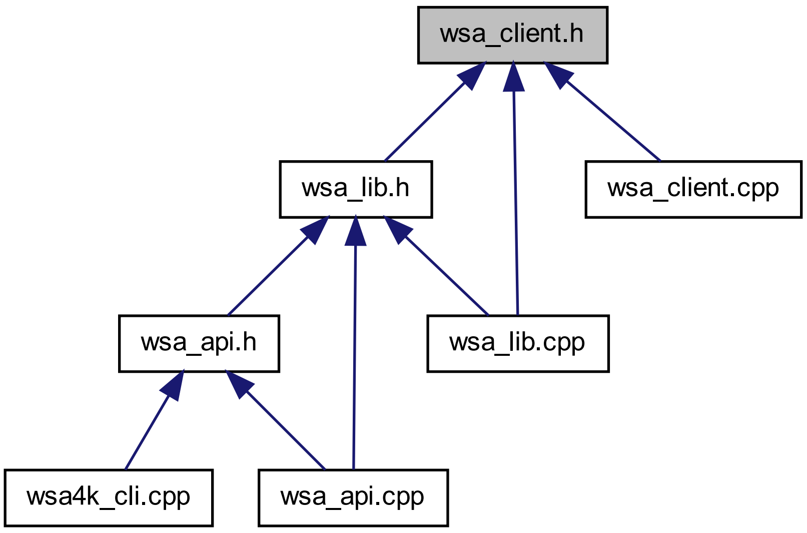
5.12.3.7 char∗ stop = "STOPDATA\0"

5.13 wsa_client.h File Reference

Include dependency graph for wsa_client.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define MAX_STR_LEN 200
- #define MAX_BUF_SIZE 20
- #define TIMEOUT 1000
- #define HISLIP 4880

Functions

- int16_t wsa_list_ips (char ∗∗ip_list)
- u_long wsa_verify_addr (const char ∗sock_addr)
- int16_t wsa_get_host_info (char ∗name)
- int16_t wsa_start_client (const char ∗wsa_addr, SOCKET ∗cmd_sock, SOCKET ∗data_sock)
- int16_t wsa_close_client (SOCKET cmd_sock, SOCKET data_sock)
- int16_t wsa_sock_send (SOCKET out_sock, char ∗out_str, int32_t len)
- int64_t wsa_sock_recv (SOCKET in_sock, char ∗rx_buf_ptr, uint32_t time_out)

5.13.1 Define Documentation

5.13.1.1 #define HISLIP 4880

5.13.1.2 #define MAX_BUF_SIZE 20

5.13.1.3 #define MAX_STR_LEN 200

5.13.1.4 #define TIMEOUT 1000

5.13.2 Function Documentation

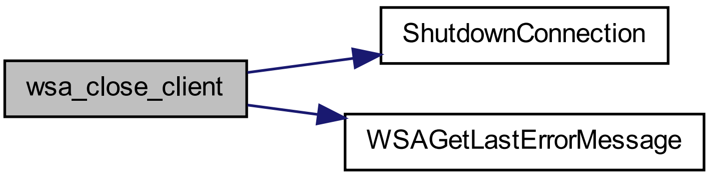
5.13.2.1 int16_t wsa_close_client (SOCKET *cmd_sock,* SOCKET *data_sock*)

Parameters

cmd_sock	-
data_sock	-

Returns

Here is the call graph for this function:



5.13.2.2 int16_t wsa_get_host_info (char ∗ *name*)

Get host information based on the name given either as IP or host name format

Parameters

name	-

Returns

5.13.2.3 int16_t wsa_list_ips (char ∗∗ *ip_list*)

Print a list of host names and the associated IP available to a user's PC.

Parameters

ip_list	-

Returns

Number of IP addresses available.

5.13.2.4 int64_t wsa_sock_recv (SOCKET *in_sock*, char ∗ *rx_buf_ptr*, uint32_t *time_out*)

Gets incoming strings from the server socket ? bytes at a time

Parameters

in_sock	-
rx_buf_ptr	-
time_out	- Time out in milliseconds

Returns

Number of "words" read

5.13.2.5 int16_t wsa_sock_send (SOCKET *out_sock*, char ∗ *out_str*, int32_t *len*)

Sends a string to the server.

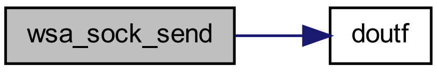
Parameters

out_sock	-
out_str	-
len	-

Returns

Number of bytes sent on success, or negative otherwise.

Here is the call graph for this function:



5.13.2.6 int16_t wsa_start_client (const char * *wsa_addr,* SOCKET * *cmd_sock,* SOCKET * *data_sock*)

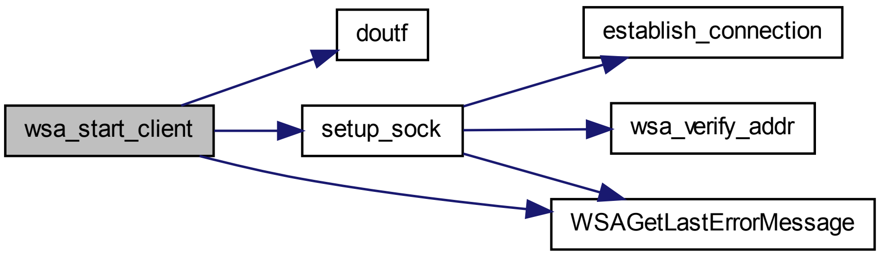
Call functions to initialize the sockets

Parameters

wsa_addr	-
cmd_sock	-
data_sock	-

Returns

Here is the call graph for this function:



5.13.2.7 u_long wsa_verify_addr (const char * *sock_addr*)

Given an address string, determine if it's a dotted-quad IP address or a domain address. If the latter, ask DNS to resolve it. In either case, return resolved IP address. If we fail, we return INADDR_NONE.

Parameters

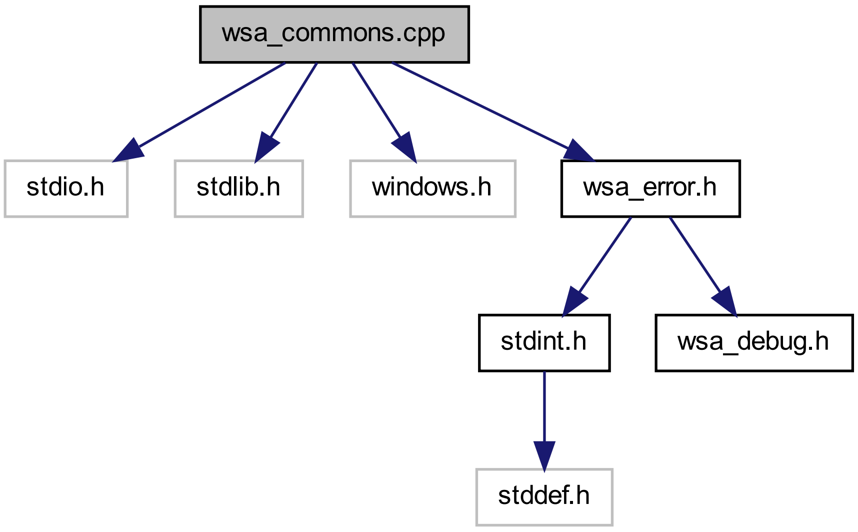
sock_addr	-

Returns

Resolved IP address or INADDR_NONE when failed.

5.14 wsa_commons.cpp File Reference

Include dependency graph for wsa_commons.cpp:



Functions

- const char ∗ wsa_get_err_msg (int16_t err_id)

5.14.1 Function Documentation

5.14.1.1 const char∗ wsa_get_err_msg (int16_t *err_id*)

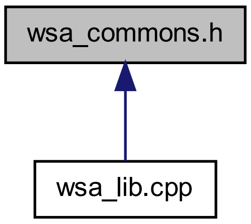
Returns the error message based on the error ID given

Parameters

err_id	The error ID

5.15 wsa_commons.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define FALSE 0
- #define TRUE 1
- #define NUM_RF_GAINS 5
- #define MHZ 1000000
- #define WSA4000 "WSA4000"
- #define WSA4000_INST_BW 125000000
- #define WSA4000_MAX_PKT_SIZE 32000
- #define WSA_RFE0560 "RFE0560"
- #define WSA_RFE0560_MAX_FREQ 7000000000
- #define WSA_RFE0560_MIN_FREQ 50
- #define WSA_RFE0560_MAX_IF_GAIN 0
- #define WSA_RFE0560_MIN_IF_GAIN -39
- #define WSA_RFE0560_FREQRES 10000
- #define WSA_RFE0560_ABS_AMP_HIGH -15
- #define WSA_RFE0560_ABS_AMP_MEDIUM 0
- #define WSA_RFE0560_ABS_AMP_LOW 13
- #define WSA_RFE0560_ABS_AMP_VLOW 20

5.15.1 Define Documentation

5.15.1.1 #define FALSE 0

5.15.1.2 #define MHZ 1000000

5.15.1.3 #define NUM_RF_GAINS 5

5.15.1.4 #define TRUE 1

5.15.1.5 #define WSA4000 "WSA4000"

5.15.1.6 #define WSA4000_INST_BW 125000000

5.15.1.7 #define WSA4000_MAX_PKT_SIZE 32000

5.15.1.8 #define WSA_RFE0560 "RFE0560"

5.15.1.9 #define WSA_RFE0560_ABS_AMP_HIGH -15

5.15.1.10 #define WSA_RFE0560_ABS_AMP_LOW 13

5.15.1.11 #define WSA_RFE0560_ABS_AMP_MEDIUM 0

5.15.1.12 #define WSA_RFE0560_ABS_AMP_VLOW 20

5.15.1.13 #define WSA_RFE0560_FREQRES 10000

5.15.1.14 #define WSA_RFE0560_MAX_FREQ 7000000000

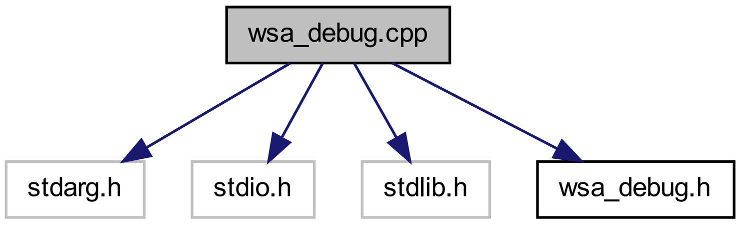
5.15.1.15 #define WSA_RFE0560_MAX_IF_GAIN 0

5.15.1.16 #define WSA_RFE0560_MIN_FREQ 50

5.15.1.17 #define WSA_RFE0560_MIN_IF_GAIN -39

5.16 wsa_debug.cpp File Reference

Include dependency graph for wsa_debug.cpp:



Functions

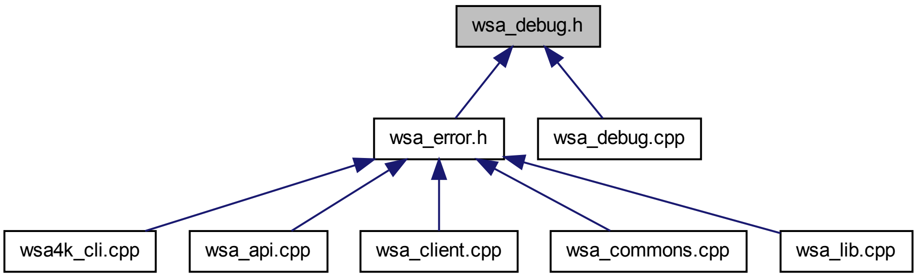
- int doutf (int level, const char ∗fmt,...)

5.16.1 Function Documentation

5.16.1.1 int doutf (int *level,* const char ∗ *fmt,* ...)

5.17 wsa_debug.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define DEBUGLEVEL 0

Functions

- int doutf (int, const char ∗,...)

5.17.1 Define Documentation

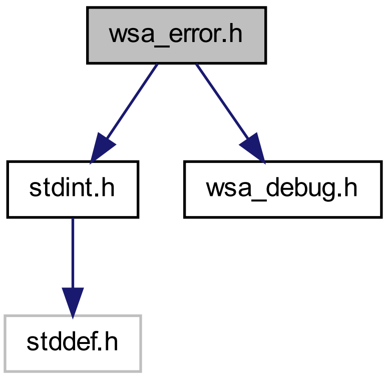
5.17.1.1 #define DEBUGLEVEL 0

5.17.2 Function Documentation

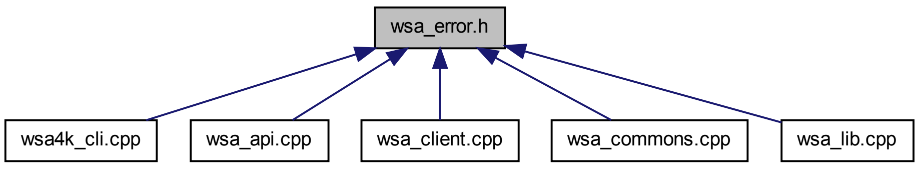
5.17.2.1 int doutf (int , const char ∗ , ...)

5.18 wsa_error.h File Reference

Include dependency graph for wsa_error.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **wsa_err_item**

Defines

- #define LNEG_NUM (-10000)
- #define WSA_ERR_NOWSA (LNEG_NUM - 1)
- #define WSA_ERR_INVIPADDRESS (LNEG_NUM - 2)
- #define WSA_ERR_NOCTRLPIPE (LNEG_NUM - 3)

- #define WSA_ERR_UNKNOWNPRODSER (LNEG_NUM - 4)
- #define WSA_ERR_UNKNOWNPRODVSN (LNEG_NUM - 5)
- #define WSA_ERR_UNKNOWNFWRVSN (LNEG_NUM - 6)
- #define WSA_ERR_UNKNOWNRFEVSN (LNEG_NUM - 7)
- #define WSA_ERR_PRODOBSOLETE (LNEG_NUM - 8)
- #define WSA_ERR_WSANOTRDY (LNEG_NUM - 101)
- #define WSA_ERR_WSAINUSE (LNEG_NUM - 102)
- #define WSA_ERR_SETFAILED (LNEG_NUM - 103)
- #define WSA_ERR_OPENFAILED (LNEG_NUM - 104)
- #define WSA_ERR_INITFAILED (LNEG_NUM - 105)
- #define WSA_ERR_INVADCCORRVALUE (LNEG_NUM - 106)
- #define WSA_ERR_INVINTFMETHOD (LNEG_NUM - 201)
- #define WSA_ERR_INVIPHOSTADDRESS (LNEG_NUM - 202)
- #define WSA_ERR_USBNOTAVBL (LNEG_NUM - 203)
- #define WSA_ERR_USBOPENFAILED (LNEG_NUM - 204)
- #define WSA_ERR_USBINITFAILED (LNEG_NUM - 205)
- #define WSA_ERR_ETHERNETNOTAVBL (LNEG_NUM - 206)
- #define WSA_ERR_ETHERNETCONNECTFAILED (LNEG_NUM - 207)
- #define WSA_ERR_ETHERNETINITFAILED (LNEG_NUM - 209)
- #define WSA_ERR_WINSOCKSTARTUPFAILED (LNEG_NUM - 210)
- #define WSA_ERR_SOCKETSETFUPFAILED (LNEG_NUM - 211)
- #define WSA_ERR_INVAMP (LNEG_NUM - 301)
- #define WSA_ERR_NODATABUS (LNEG_NUM - 401)
- #define WSA_ERR_READFRAMEFAILED (LNEG_NUM - 402)
- #define WSA_ERR_INVSAMPLESIZE (LNEG_NUM - 403)
- #define WSA_ERR_FREQOUTOFBOUND (LNEG_NUM - 601)
- #define WSA_ERR_INVFREQRES (LNEG_NUM - 602)
- #define WSA_ERR_FREQSETFAILED (LNEG_NUM - 603)
- #define WSA_ERR_PLLLOCKFAILED (LNEG_NUM - 604)
- #define WSA_ERR_INVRFGAIN (LNEG_NUM - 801)
- #define WSA_ERR_INVIFGAIN (LNEG_NUM - 802)
- #define WSA_ERR_IFGAINSETFAILED (LNEG_NUM - 803)
- #define WSA_ERR_RFGAINSETFAILED (LNEG_NUM - 804)
- #define WSA_ERR_INVRUNMODE (LNEG_NUM - 1001)
- #define WSA_ERR_INVTRIGID (LNEG_NUM - 1201)
- #define WSA_ERR_INVSTOPFREQ (LNEG_NUM - 1202)
- #define WSA_ERR_STARTOOB (LNEG_NUM - 1203)
- #define WSA_ERR_STOPOOB (LNEG_NUM - 1204)
- #define WSA_ERR_INVSTARTRES (LNEG_NUM - 1205)
- #define WSA_ERR_INVSTOPRES (LNEG_NUM - 1206)
- #define WSA_ERR_INVTRIGRANGE (LNEG_NUM - 1207)
- #define WSA_ERR_INVDWELL (LNEG_NUM - 1208)
- #define WSA_ERR_INVNUMFRAMES (LNEG_NUM - 1209)
- #define WSA_ERR_CMDSENDFAILED (LNEG_NUM - 1501)
- #define WSA_ERR_CMDINVALID (LNEG_NUM - 1502)
- #define WSA_ERR_INVANTENNAPORT (LNEG_NUM - 1601)

- #define WSA_ERR_ANTENNASETFAILED (LNEG_NUM - 1602)
- #define WSA_ERR_INVFILTERMODE (LNEG_NUM - 1603)
- #define WSA_ERR_FILTERSETFAILED (LNEG_NUM - 1604)
- #define WSA_ERR_INVCALIBRATEMODE (LNEG_NUM - 1605)
- #define WSA_ERR_CALIBRATESETFAILED (LNEG_NUM - 1606)
- #define WSA_ERR_FILECREATEFAILED (LNEG_NUM - 1900)
- #define WSA_ERR_FILEOPENFAILED (LNEG_NUM - 1901)
- #define WSA_ERR_FILEREADFAILED (LNEG_NUM - 1902)
- #define WSA_ERR_FILEWRITEFAILED (LNEG_NUM - 1903)
- #define WSA_ERR_INVNUMBER (LNEG_NUM - 2000)
- #define WSA_ERR_INVREGADDR (LNEG_NUM - 2001)
- #define WSA_ERR_MALLOCFAILED (LNEG_NUM - 2002)
- #define WSA_ERR_UNKNOWN_ERROR (LNEG_NUM - 2003)

Functions

- const char ∗ wsa_get_err_msg (int16_t err_id)

5.18.1 Define Documentation

5.18.1.1 #define LNEG_NUM (-10000)

5.18.1.2 #define WSA_ERR_ANTENNASETFAILED (LNEG_NUM - 1602)

5.18.1.3 #define WSA_ERR_CALIBRATESETFAILED (LNEG_NUM - 1606)

5.18.1.4 #define WSA_ERR_CMDINVALID (LNEG_NUM - 1502)

5.18.1.5 #define WSA_ERR_CMDSENDFAILED (LNEG_NUM - 1501)

5.18.1.6 #define WSA_ERR_ETHERNETCONNECTFAILED (LNEG_NUM - 207)

5.18.1.7 #define WSA_ERR_ETHERNETINITFAILED (LNEG_NUM - 209)

5.18.1.8 #define WSA_ERR_ETHERNETNOTAVBL (LNEG_NUM - 206)

5.18.1.9 #define WSA_ERR_FILECREATEFAILED (LNEG_NUM - 1900)

5.18.1.10 #define WSA_ERR_FILEOPENFAILED (LNEG_NUM - 1901)

5.18.1.11 #define WSA_ERR_FILEREADFAILED (LNEG_NUM - 1902)

5.18.1.12 #define WSA_ERR_FILEWRITEFAILED (LNEG_NUM - 1903)

5.18.1.13 #define WSA_ERR_FILTERSETFAILED (LNEG_NUM - 1604)

5.18.1.14 #define WSA_ERR_FREQOUTOFBOUND (LNEG_NUM - 601)

5.18.1.15 #define WSA_ERR_FREQSETFAILED (LNEG_NUM - 603)

5.18.1.16 #define WSA_ERR_IFGAINSETFAILED (LNEG_NUM - 803)

5.18.1.17 #define WSA_ERR_INITFAILED (LNEG_NUM - 105)

5.18.1.18 #define WSA_ERR_INVADCCORRVALUE (LNEG_NUM - 106)

5.18.1.19 #define WSA_ERR_INVAMP (LNEG_NUM - 301)

5.18.1.20 #define WSA_ERR_INVANTENNAPORT (LNEG_NUM - 1601)

5.18.1.21 #define WSA_ERR_INVCALIBRATEMODE (LNEG_NUM - 1605)

5.18.1.22 #define WSA_ERR_INVDWELL (LNEG_NUM - 1208)

5.18.1.23 #define WSA_ERR_INVFILTERMODE (LNEG_NUM - 1603)

5.18.1.24 #define WSA_ERR_INVFREQRES (LNEG_NUM - 602)

5.18.1.25 #define WSA_ERR_INVIFGAIN (LNEG_NUM - 802)

5.18.1.26 #define WSA_ERR_INVINTFMETHOD (LNEG_NUM - 201)

5.18.1.27 #define WSA_ERR_INVIPADDRESS (LNEG_NUM - 2)

5.18.1.28 #define WSA_ERR_INVIPHOSTADDRESS (LNEG_NUM - 202)

5.18.1.29 #define WSA_ERR_INVNUMBER (LNEG_NUM - 2000)

5.18.1.30 #define WSA_ERR_INVNUMFRAMES (LNEG_NUM - 1209)

5.18.1.31 #define WSA_ERR_INVREGADDR (LNEG_NUM - 2001)

5.18.1.32 #define WSA_ERR_INVRFGAIN (LNEG_NUM - 801)

5.18.1.33 #define WSA_ERR_INVRUNMODE (LNEG_NUM - 1001)

5.18.1.34 #define WSA_ERR_INVSAMPLESIZE (LNEG_NUM - 403)

5.18.1.35 #define WSA_ERR_INVSTARTRES (LNEG_NUM - 1205)

5.18.1.36 #define WSA_ERR_INVSTOPFREQ (LNEG_NUM - 1202)

5.18.1.37 #define WSA_ERR_INVSTOPRES (LNEG_NUM - 1206)

5.18.1.38 #define WSA_ERR_INVTRIGID (LNEG_NUM - 1201)

5.18.1.39 #define WSA_ERR_INVTRIGRANGE (LNEG_NUM - 1207)

5.18.1.40 #define WSA_ERR_MALLOCFAILED (LNEG_NUM - 2002)

5.18.1.41 #define WSA_ERR_NOCTRLPIPE (LNEG_NUM - 3)

5.18.1.42 #define WSA_ERR_NODATABUS (LNEG_NUM - 401)

5.18.1.43 #define WSA_ERR_NOWSA (LNEG_NUM - 1)

5.18.1.44 #define WSA_ERR_OPENFAILED (LNEG_NUM - 104)

5.18.1.45 #define WSA_ERR_PLLLOCKFAILED (LNEG_NUM - 604)

5.18.1.46 #define WSA_ERR_PRODOBSOLETE (LNEG_NUM - 8)

5.18.1.47 #define WSA_ERR_READFRAMEFAILED (LNEG_NUM - 402)

5.18.1.48 #define WSA_ERR_RFGAINSETFAILED (LNEG_NUM - 804)

5.18.1.49 #define WSA_ERR_SETFAILED (LNEG_NUM - 103)

5.18.1.50 #define WSA_ERR_SOCKETSETFUPFAILED (LNEG_NUM - 211)

5.18.1.51 #define WSA_ERR_STARTOOB (LNEG_NUM - 1203)

5.18.1.52 #define WSA_ERR_STOPOOB (LNEG_NUM - 1204)

5.18.1.53 #define WSA_ERR_UNKNOWN_ERROR (LNEG_NUM - 2003)

5.18.1.54 #define WSA_ERR_UNKNOWNFWRVSN (LNEG_NUM - 6)

5.18.1.55 #define WSA_ERR_UNKNOWNPRODSER (LNEG_NUM - 4)

5.18.1.56 #define WSA_ERR_UNKNOWNPRODVSN (LNEG_NUM - 5)

5.18.1.57 #define WSA_ERR_UNKNOWNRFEVSN (LNEG_NUM - 7)

5.18.1.58 #define WSA_ERR_USBINITFAILED (LNEG_NUM - 205)

5.18.1.59 #define WSA_ERR_USBNOTAVBL (LNEG_NUM - 203)

5.18.1.60 #define WSA_ERR_USBOPENFAILED (LNEG_NUM - 204)

5.18.1.61 #define WSA_ERR_WINSOCKSTARTUPFAILED (LNEG_NUM - 210)

5.18.1.62 #define WSA_ERR_WSAINUSE (LNEG_NUM - 102)

5.18.1.63 #define WSA_ERR_WSANOTRDY (LNEG_NUM - 101)

5.18.2 Function Documentation

5.18.2.1 const char∗ wsa_get_err_msg (int16_t *err_id*)

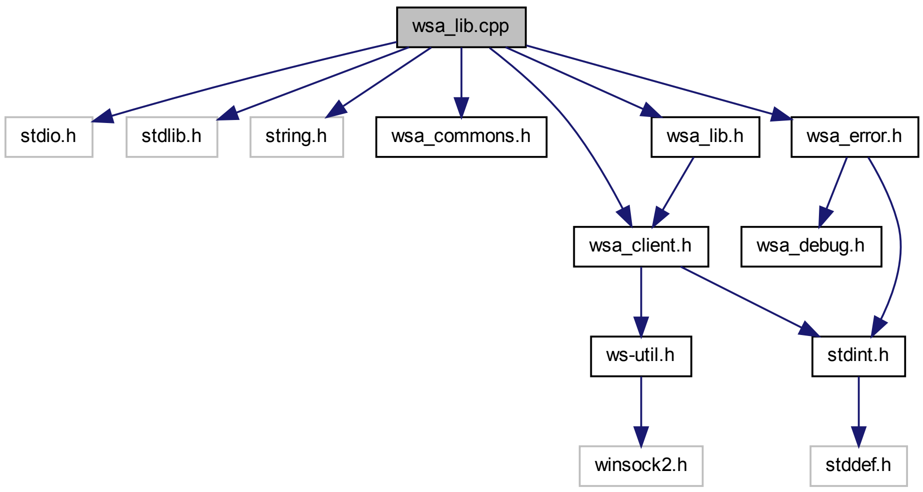
Returns the error message based on the error ID given

Parameters

err_id	The error ID

5.19 wsa_lib.cpp File Reference

Include dependency graph for wsa_lib.cpp:



Defines

- #define MAX_FILE_LINES 300
- #define SEP_CHARS "\n\r"

Functions

- int16_t wsa_tokenize_file (FILE ∗fptr, char ∗cmd_str[ ])
- int16_t wsa_dev_init (struct wsa_device ∗dev)
- int16_t wsa_connect (struct wsa_device ∗dev, char ∗cmd_syntax, char ∗intf_-method)
- int16_t wsa_disconnect (struct wsa_device ∗dev)
- int16_t wsa_list_devs (char ∗∗wsa_list)
- int16_t wsa_send_command (struct wsa_device ∗dev, char ∗command)
- int16_t wsa_send_command_file (struct wsa_device ∗dev, char ∗file_name)
- struct wsa_resp wsa_send_query (struct wsa_device ∗dev, char ∗command)
- int16_t wsa_query_error (struct wsa_device ∗dev)
- int64_t wsa_get_frame (struct wsa_device ∗dev, struct wsa_frame_header ∗header, int32_t ∗i_buf, int32_t ∗q_buf, uint64_t sample_size)

5.19.1 Define Documentation

5.19.1.1 #define MAX_FILE_LINES 300

5.19.1.2 #define SEP_CHARS "\n\r"

5.19.2 Function Documentation

5.19.2.1 int16_t wsa_connect (struct wsa_device ∗ *dev,* char ∗ *cmd_syntax,* char ∗ *intf_method*)

Connect to a WSA through the specified interface method **intf_method**, and communicate control commands in the format of the given command syntax.

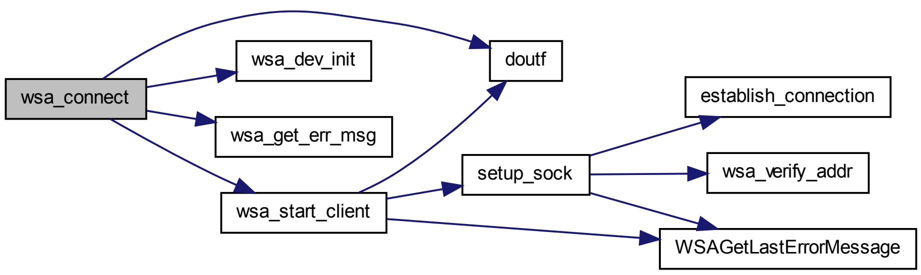
Parameters

dev	- A pointer to the WSA device structure to be connected/establised.
cmd_syntax	- A char pointer to store standard for control commands communication to the WSA. Currently supported standard command syntax type is: SCPI.
intf_method	- A char pointer to store the interface method to the WSA. Possible methods: • With LAN, use: "TCPIP::<Ip address of the WSA>::HISLIP" • With USB, use: "USB" (check if supported with the WSA version used)

Returns

0 on success, or a negative number on error. TODO: define ERROR values with associated messages....

Here is the call graph for this function:



5.19.2.2 int16_t wsa_dev_init (struct wsa_device * *dev*)

Initialized the the wsa_device structure

Parameters

dev	- A pointer to the WSA device structure.

Returns

None

5.19.2.3 int16_t wsa_disconnect (struct wsa_device * *dev*)

Close the device connection if one is started, stop any existing data capture, and perform any necessary clean ups.

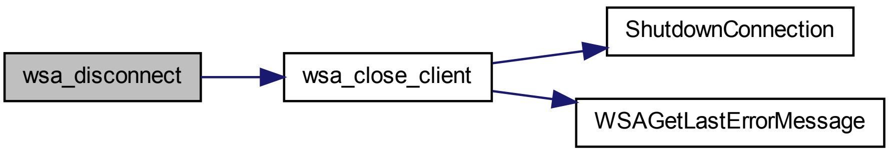
Parameters

dev	- A pointer to the WSA device structure to be closed.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.19.2.4 int64_t wsa_get_frame (struct wsa_device * *dev*, struct wsa_frame_header * *header*, int32_t * *i_buf*, int32_t * *q_buf*, uint64_t *sample_size*)

Reads a frame of data. *Each* frame consists of a header, and I and Q buffers of data of length determine by the **sample_size** parameter.

Parameters

dev	- A pointer to the WSA device structure.
header	- A pointer to **wsa_frame_header** structure to store information for the frame.

i_buf	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the sample_size.
q_buf	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the sample_size.
sample_size	- A 64-bit unsigned integer sample size (i.e. {I, Q} sample pairs) per data frame to be captured. The frame size is limited to a maximum number, **max_sample_size**, listed in the **wsa_descriptor** structure.

Returns

Number of samples read on success, or a negative number on error.

5.19.2.5 int16_t wsa_list_devs (char ∗∗ *wsa_list*)

List (print out) the IPs of connected WSAs to the network? or the PC??? For now, will list the IPs for any of the connected devices to a PC?

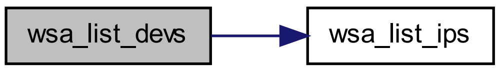
Parameters

wsa_list	- A double char pointer to store (WSA???) IP addresses connected to a network???.

Returns

Number of connected WSAs (or IPs for now) on success, or a negative number on error.

Here is the call graph for this function:



5.19.2.6 int16_t wsa_query_error (struct wsa_device ∗ *dev*)

Querry the WSA for any error.

Parameters

dev	- A pointer to the WSA device structure.

Returns

0 on success, or a negative number on error.

5.19.2.7 int16_t wsa_send_command (struct wsa_device * *dev*, char * *command*)

Open a file or print the help commands information associated with the WSA used.

Parameters

dev	- The WSA device structure from which the help information will be provided.

Returns

0 on success, or a negative number on error. Send the control command string to the WSA device specified by **dev**. The commands format must be written according to the specified standard syntax in wsa_connect().

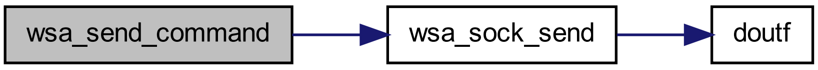
Parameters

dev	- A pointer to the WSA device structure.
command	- A char pointer to the control command string written in the format specified by the syntax standard in wsa_connect()

Returns

Number of bytes sent on success, or a negative number on error.

Here is the call graph for this function:



5.19.2.8 int16_t wsa_send_command_file (struct wsa_device * *dev*, char * *file_name*)

Read command line(s) stored in the given **file_name** and send each line to the WSA.

Remarks

- Assuming each command line is for a single function followed by a new line.
- Currently read only SCPI commands. Other types of commands, TBD.

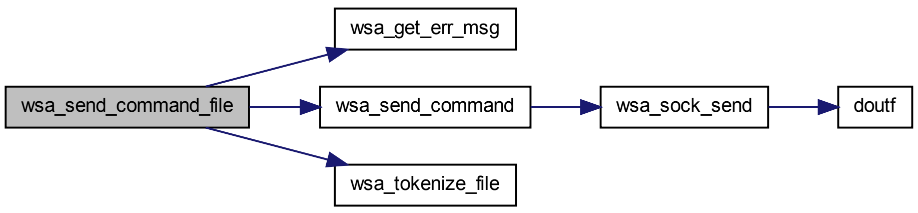
Parameters

dev	- A pointer to the WSA device structure.
file_name	- A pointer to the file name

Returns

Number of command lines at success, or a negative error number.

Here is the call graph for this function:



5.19.2.9 struct wsa_resp wsa_send_query (struct wsa_device * *dev,* char * *command*) `[read]`

Send query command to the WSA device specified by **dev**. The commands format must be written according to the specified command syntax in wsa_connect().

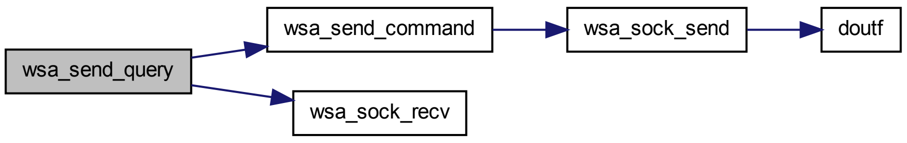
Parameters

dev	- A pointer to the WSA device structure.
command	- A char pointer to the query command string written in the format specified by the command syntax in wsa_connect().

Returns

The result stored in a wsa_resp struct format.

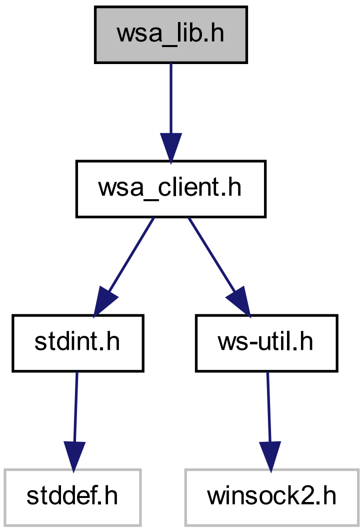
Here is the call graph for this function:



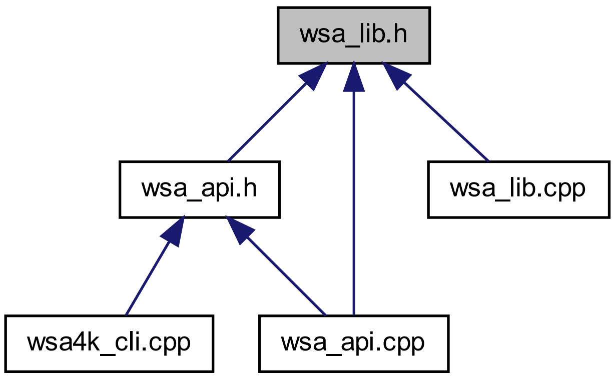
5.19.2.10 int16_t wsa_tokenize_file (FILE * *fptr,* char * *cmd_str[]*)

5.20 wsa_lib.h File Reference

Include dependency graph for wsa_lib.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct wsa_descriptor

    This structure stores WSA information.
- struct wsa_time

    This structure contains the time information. It is used for the time stamp in a frame header.
- struct wsa_frame_header

    This structure contains header information related to each frame read by wsa_get_-frame().
- struct wsa_socket

    A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition.
- struct wsa_device

    A structure containing the components associate with each WSA device.
- struct wsa_resp

    This structure contains the response information for each query.

Defines

- #define FALSE 0
- #define TRUE 1
- #define NUM_RF_GAINS 5
- #define SCPI "SCPI"

Enumerations

- enum wsa_gain {
  WSA_GAIN_HIGH = 1, WSA_GAIN_MEDIUM, WSA_GAIN_LOW, WSA_GAIN_-VLOW,
  WSA_GAIN_HIGH = 1, WSA_GAIN_MEDIUM, WSA_GAIN_LOW, WSA_GAIN_-VLOW }

Functions

- int16_t wsa_connect (struct wsa_device ∗dev, char ∗cmd_syntax, char ∗intf_-method)
- int16_t wsa_disconnect (struct wsa_device ∗dev)
- int16_t wsa_list_devs (char ∗∗wsa_list)
- int16_t wsa_send_command (struct wsa_device ∗dev, char ∗command)
- int16_t wsa_send_command_file (struct wsa_device ∗dev, char ∗file_name)
- struct wsa_resp wsa_send_query (struct wsa_device ∗dev, char ∗command)
- int16_t wsa_query_error (struct wsa_device ∗dev)
- int64_t wsa_get_frame (struct wsa_device ∗dev, struct wsa_frame_header ∗header, int32_t ∗i_buf, int32_t ∗q_buf, uint64_t sample_size)

5.20.1 Define Documentation

5.20.1.1 #define FALSE 0

5.20.1.2 #define NUM_RF_GAINS 5

5.20.1.3 #define SCPI "SCPI"

5.20.1.4 #define TRUE 1

5.20.2 Enumeration Type Documentation

5.20.2.1 enum wsa_gain

Enumerator:

WSA_GAIN_HIGH High RF amplification. Value 1.

WSA_GAIN_MEDIUM Medium RF amplification.

WSA_GAIN_LOW Low RF amplification.

WSA_GAIN_VLOW Very low RF amplification.

WSA_GAIN_HIGH

WSA_GAIN_MEDIUM

WSA_GAIN_LOW

WSA_GAIN_VLOW

5.20.3 Function Documentation

5.20.3.1 int16_t wsa_connect (struct wsa_device ∗ *dev*, char ∗ *cmd_syntax*, char ∗ *intf_method*)

Connect to a WSA through the specified interface method **intf_method**, and communicate control commands in the format of the given command syntax.

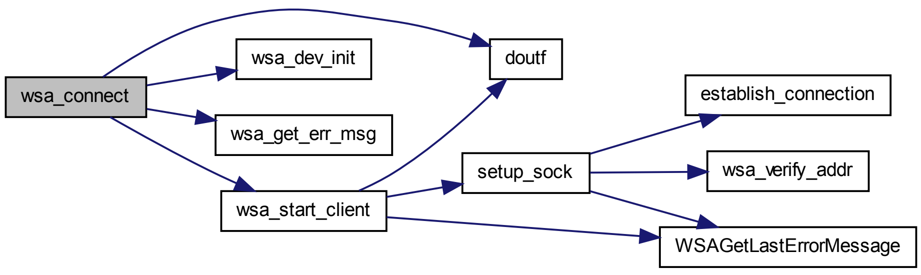
Parameters

dev	- A pointer to the WSA device structure to be connected/establised.
cmd_syntax	- A char pointer to store standard for control commands communication to the WSA. Currently supported standard command syntax type is: SCPI.
intf_method	- A char pointer to store the interface method to the WSA. Possible methods: • With LAN, use: "TCPIP::<Ip address of the WSA>::HISLIP" • With USB, use: "USB" (check if supported with the WSA version used)

Returns

0 on success, or a negative number on error. TODO: define ERROR values with associated messages....

Here is the call graph for this function:



5.20.3.2 int16_t wsa_disconnect (struct wsa_device ∗ *dev*)

Close the device connection if one is started, stop any existing data capture, and perform any necessary clean ups.

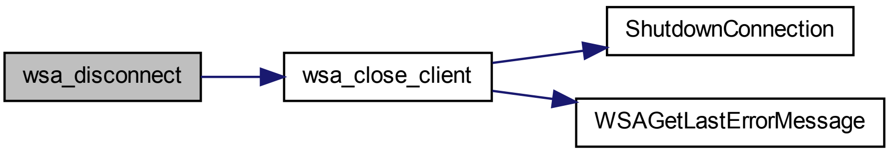
Parameters

dev	- A pointer to the WSA device structure to be closed.

Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



5.20.3.3 int64_t wsa_get_frame (struct wsa_device * *dev*, struct wsa_frame_header * *header*, int32_t * *i_buf*, int32_t * *q_buf*, uint64_t *sample_size*)

Reads a frame of data. *Each* frame consists of a header, and I and Q buffers of data of length determine by the **sample_size** parameter.

Parameters

dev	- A pointer to the WSA device structure.
header	- A pointer to **wsa_frame_header** structure to store information for the frame.
i_buf	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the sample_size.
q_buf	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the sample_size.
sample_size	- A 64-bit unsigned integer sample size (i.e. {I, Q} sample pairs) per data frame to be captured. The frame size is limited to a maximum number, **max_sample_size**, listed in the **wsa_descriptor** structure.

Returns

Number of samples read on success, or a negative number on error.

5.20.3.4 int16_t wsa_list_devs (char ** *wsa_list*)

List (print out) the IPs of connected WSAs to the network? or the PC??? For now, will list the IPs for any of the connected devices to a PC?

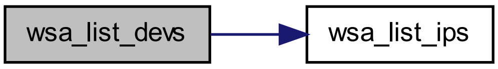
Parameters

wsa_list	- A double char pointer to store (WSA???) IP addresses connected to a network???.

Returns

Number of connected WSAs (or IPs for now) on success, or a negative number on error.

Here is the call graph for this function:



5.20.3.5 int16_t wsa_query_error (struct wsa_device * *dev*)

Querry the WSA for any error.

Parameters

dev	- A pointer to the WSA device structure.

Returns

0 on success, or a negative number on error.

5.20.3.6 int16_t wsa_send_command (struct wsa_device * *dev*, char * *command*)

Open a file or print the help commands information associated with the WSA used.

Parameters

dev	- The WSA device structure from which the help information will be provided.

Returns

0 on success, or a negative number on error. Send the control command string to the WSA device specified by **dev**. The commands format must be written according to the specified standard syntax in wsa_connect().

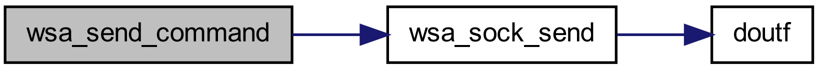
Parameters

dev	- A pointer to the WSA device structure.
command	- A char pointer to the control command string written in the format specified by the syntax standard in wsa_connect()

Returns

Number of bytes sent on success, or a negative number on error.

Here is the call graph for this function:



5.20.3.7 int16_t wsa_send_command_file (struct wsa_device ∗ *dev*, char ∗ *file_name*)

Read command line(s) stored in the given **file_name** and send each line to the WSA.

Remarks

- Assuming each command line is for a single function followed by a new line.
- Currently read only SCPI commands. Other types of commands, TBD.

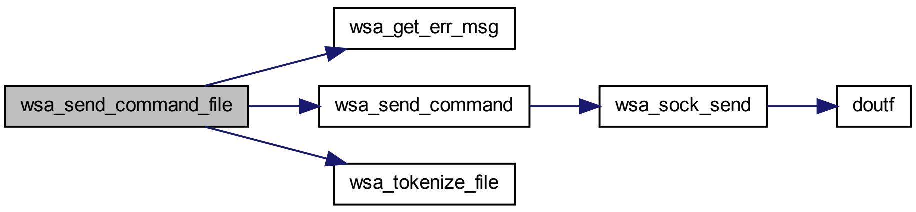
Parameters

dev	- A pointer to the WSA device structure.
file_name	- A pointer to the file name

Returns

Number of command lines at success, or a negative error number.

Here is the call graph for this function:



5.20.3.8 struct **wsa_resp** wsa_send_query (struct **wsa_device** ∗ *dev*, char ∗ *command*) [read]

Send query command to the WSA device specified by **dev**. The commands format must be written according to the specified command syntax in wsa_connect().

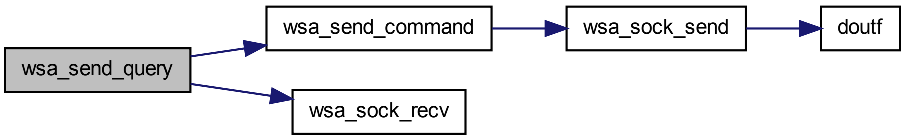
Parameters

dev	- A pointer to the WSA device structure.
command	- A char pointer to the query command string written in the format specified by the command syntax in wsa_connect().

Returns

The result stored in a wsa_resp struct format.

Here is the call graph for this function:



5.21 wsa_lib.txt File Reference

Contain some code documents for wsa_lib.h.

5.21.1 Detailed Description

Index