

WSA4000 CLI (Command Line Interface) Program Documentation

Generated by Doxygen 1.7.4

Mon Aug 29 2011 15:48:21

Contents

1	Introduction	1
1.1	Limitations in v1.0:	1
2	Data Structure Index	2
2.1	Data Structures	2
3	File Index	2
3.1	File List	2
4	Data Structure Documentation	3
4.1	wsa_descriptor Struct Reference	3
4.1.1	Field Documentation	4
4.2	wsa_device Struct Reference	4
4.2.1	Field Documentation	5
4.3	wsa_frame_header Struct Reference	5
4.3.1	Field Documentation	6
4.4	wsa_resp Struct Reference	7
4.4.1	Field Documentation	7
4.5	wsa_socket Struct Reference	7
4.5.1	Field Documentation	7
4.6	wsa_time Struct Reference	7
4.6.1	Field Documentation	8
5	File Documentation	8
5.1	main.cpp File Reference	8
5.1.1	Function Documentation	9
5.2	ReadMe.txt File Reference	9
5.2.1	Variable Documentation	9
5.3	stdint.h File Reference	10
5.3.1	Typedef Documentation	11
5.4	targetver.h File Reference	11
5.4.1	Define Documentation	11
5.5	ws-util.cpp File Reference	12
5.5.1	Function Documentation	12

5.5.2	Variable Documentation	12
5.6	ws-util.h File Reference	13
5.6.1	Function Documentation	14
5.7	wsa4k_cli.cpp File Reference	15
5.7.1	Function Documentation	15
5.8	wsa4k_cli.h File Reference	18
5.8.1	Define Documentation	19
5.8.2	Function Documentation	19
5.8.3	Variable Documentation	20
5.9	wsa_api.cpp File Reference	21
5.9.1	Function Documentation	22
5.10	wsa_api.h File Reference	30
5.10.1	Enumeration Type Documentation	32
5.10.2	Function Documentation	32
5.11	wsa_client.cpp File Reference	40
5.11.1	Define Documentation	41
5.11.2	Function Documentation	41
5.11.3	Variable Documentation	45
5.12	wsa_client.h File Reference	46
5.12.1	Define Documentation	47
5.12.2	Function Documentation	47
5.13	wsa_commons.cpp File Reference	50
5.13.1	Function Documentation	50
5.14	wsa_commons.h File Reference	51
5.14.1	Define Documentation	52
5.15	wsa_debug.cpp File Reference	52
5.15.1	Function Documentation	53
5.16	wsa_debug.h File Reference	53
5.16.1	Define Documentation	53
5.16.2	Function Documentation	53
5.17	wsa_error.h File Reference	54
5.17.1	Define Documentation	56
5.17.2	Function Documentation	57
5.18	wsa_lib.cpp File Reference	58

5.18.1 Function Documentation	58
5.19 wsa_lib.h File Reference	63
5.19.1 Define Documentation	65
5.19.2 Enumeration Type Documentation	65
5.19.3 Function Documentation	65
5.20 wsa_lib.txt File Reference	70
5.20.1 Detailed Description	70

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.

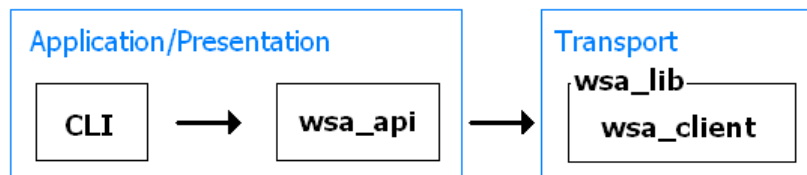


Figure 1: Interface Layers to WSA on PC Side

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, query 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 calibration. 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:

wsa_descriptor (This structure stores WSA information)	3
wsa_device (A structure containing the components associate with each WSA device)	4
wsa_frame_header (This structure contains header information related to each frame read by wsa_get_frame())	5
wsa_resp (This structure contains the response information for each query)	7
wsa_socket (A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition)	7
wsa_time (This structure contains the time information. It is used for the time stamp in a frame header)	7

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

main.cpp	8
---------------------------------	----------

stdint.h	10
targetver.h	11
ws-util.cpp	12
ws-util.h	13
wsa4k_cli.cpp	15
wsa4k_cli.h	18
wsa_api.cpp	21
wsa_api.h	30
wsa_client.cpp	40
wsa_client.h	46
wsa_commons.cpp	50
wsa_commons.h	51
wsa_debug.cpp	52
wsa_debug.h	53
wsa_error.h	54
wsa_lib.cpp	58
wsa_lib.h	63

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](#) [20]
- 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](#)

4.1.1 Field Documentation

4.1.1.1 char fw_version

The firmware version currently in the WSA.

4.1.1.2 uint64_t inst_bw

The WSA instantaneous bandwidth in Hz.

4.1.1.3 char intf_type

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

4.1.1.4 uint64_t max_sample_size

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

4.1.1.5 uint64_t max_tune_freq

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

4.1.1.6 uint64_t min_tune_freq

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

4.1.1.7 char prod_name

WSA product name.

4.1.1.8 char prod_serial

WSA product serial number.

4.1.1.9 char prod_version

WSA product version number.

4.1.1.10 char rfe_name

WSA product name.

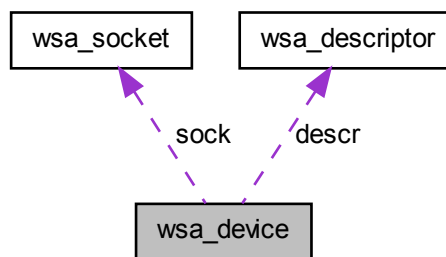
4.1.1.11 char rfe_version

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](#).

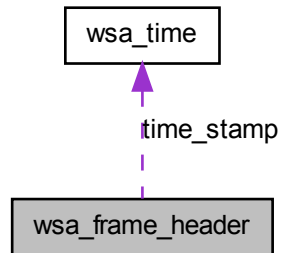
4.2.1.2 struct `wsa_socket` `sock`

The socket structure component of the WSA, used for TCP/IP 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](#)

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).

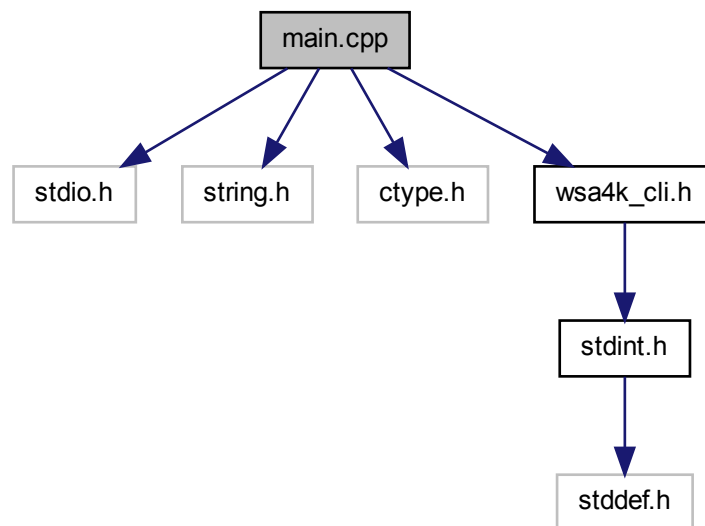
4.6.1.2 int32_t sec

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

5 File Documentation

5.1 main.cpp File Reference

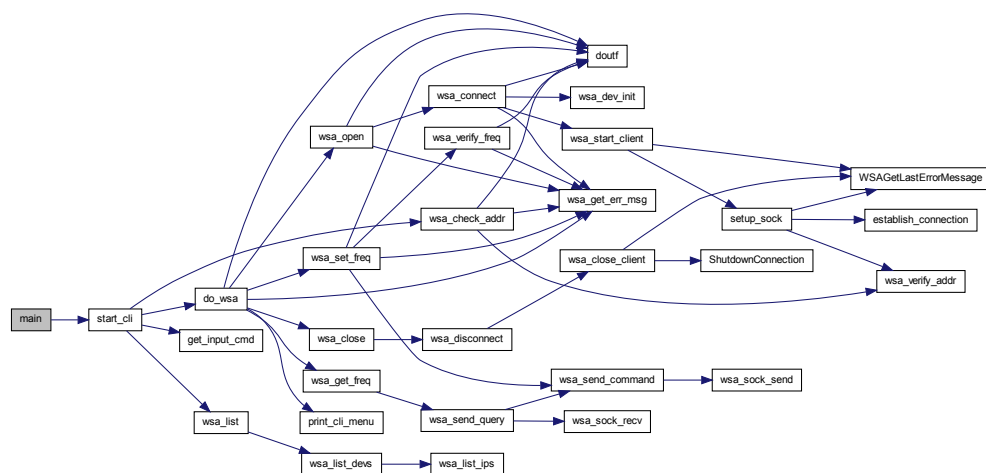
Include dependency graph for main.cpp:



- `int32_t main (int32_t argc, char *argv[])`

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

Here is the call graph for this function:



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](#)

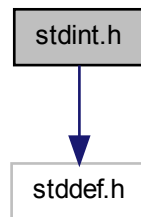
5.2.1.1 and information about the configurations

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

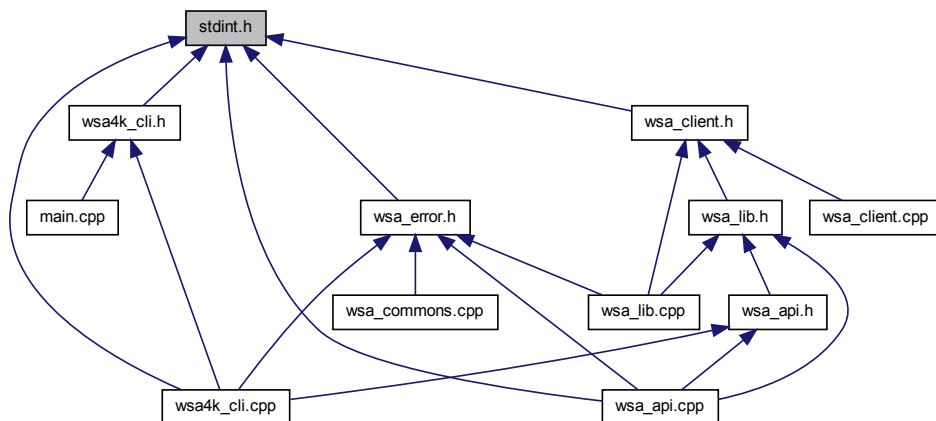
5.2.1.3 and information about the platforms

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

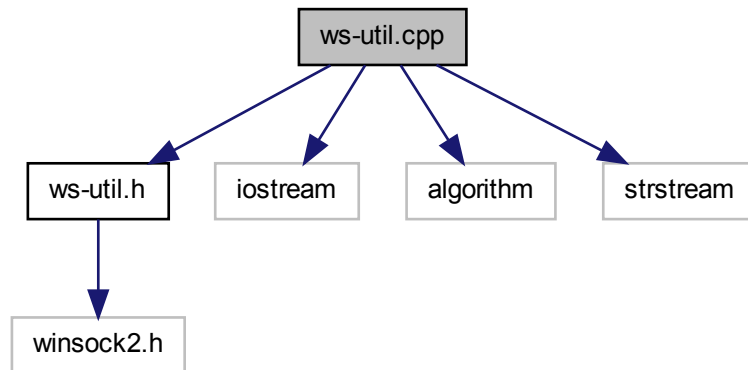
- #define [_WIN32_WINNT](#) 0x0600

5.4.1 Define Documentation

5.4.1.1 #define _WIN32_WINNT 0x0600

5.5 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)

5.5.1 Function Documentation

5.5.1.1 bool [ShutdownConnection](#) (SOCKET *sd*, char * *sock_name*)

5.5.1.2 const char* [WSAGetLastErrorMessage](#) (const char * *pcMessagePrefix*, int *nErrorID*)

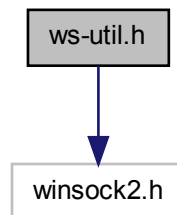
5.5.2 Variable Documentation

5.5.2.1 `const int kBufferSize = 1024`

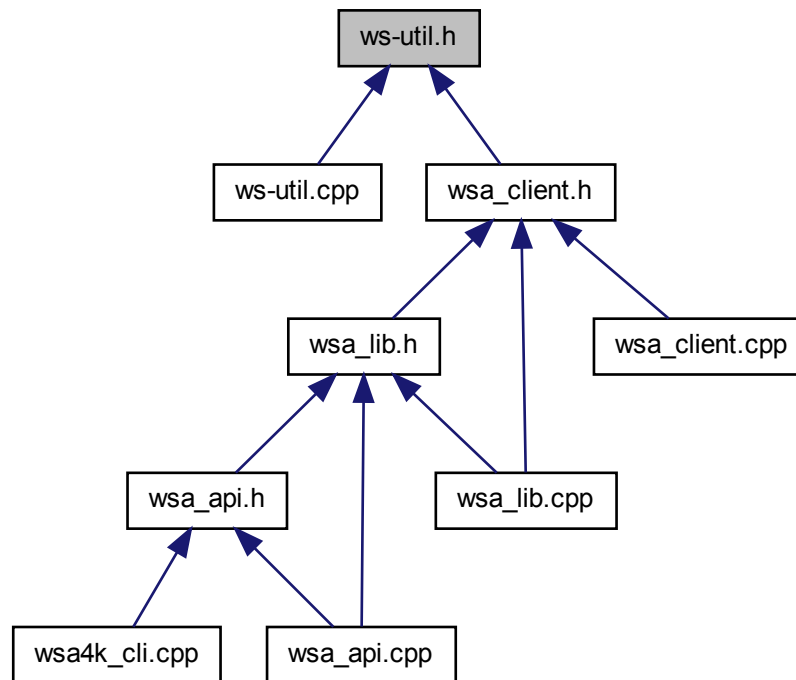
5.5.2.2 `const int kNumMessages = sizeof(gaErrorList) / sizeof(ErrorEntry)`

5.6 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)

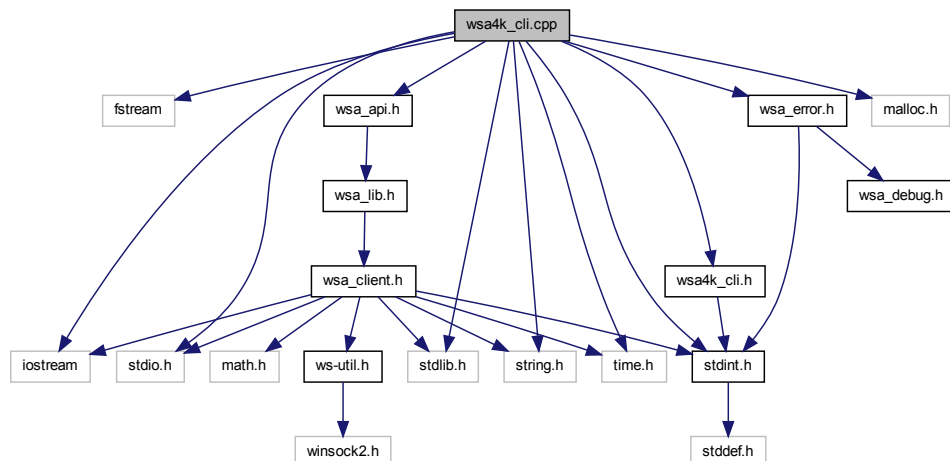
5.6.1 Function Documentation

5.6.1.1 bool [ShutdownConnection](#) (SOCKET *sd*, char * *sock_name*)

5.6.1.2 const char* [WSAGetLastErrorMessage](#) (const char * *pcMessagePrefix*, int *nErrorID* = 0)

5.7 wsa4k_cli.cpp File Reference

Include dependency graph for wsa4k_cli.cpp:



Functions

- void [print_cli_menu](#) (struct [wsa_device](#) *dev)
- char * [get_input_cmd](#) (uint8_t pretext)
- int16_t [do_wsa](#) (const char *wsa_addr)
- int16_t [start_cli](#) (void)

5.7.1 Function Documentation

5.7.1.1 int16_t do_wsa (const char * wsa_addr)

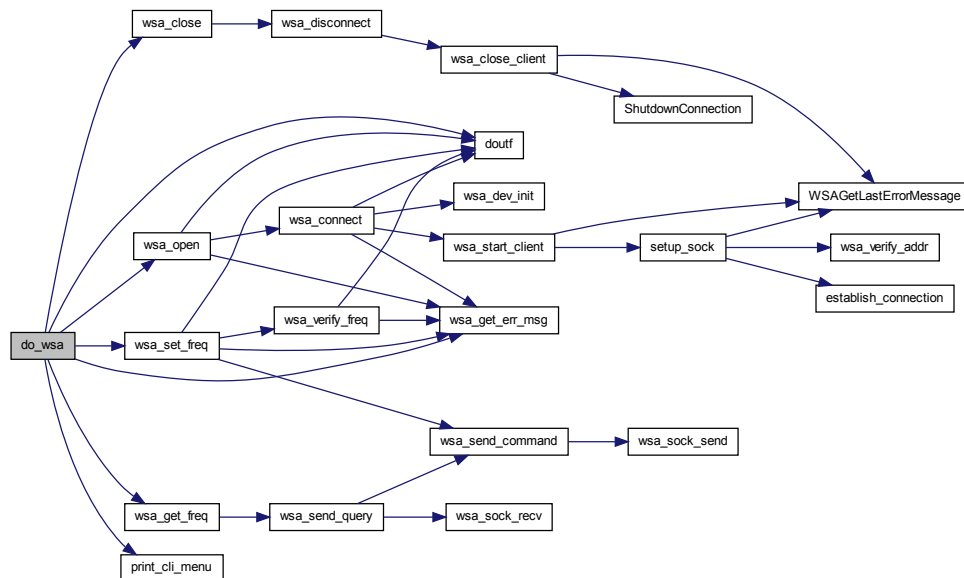
Setup WSA device variables, start the WSA connection and

Parameters

<i>wsa_addr</i>

Returns

Here is the call graph for this function:



5.7.1.2 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

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

Returns

The characters inputted.

5.7.1.3 void print_cli_menu (struct wsa_device * dev)

Print out the CLI options menu

Parameters

<i>dev</i>	- a wsa device structure.
------------	---------------------------

Returns

None

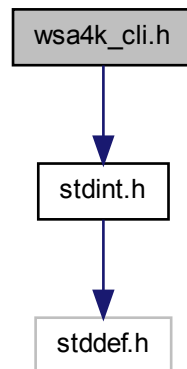
Start the CLI tool. First get a valid IP address from users, verify and start the WSA connection.

0 if successful

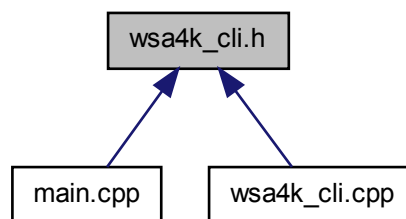
[illegible]

5.8 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_STR_LEN` 200
- #define `MAX_BUF_SIZE` 20
- #define `MAX_FS` 1000
- #define `MHZ` 1000000
- #define `FALSE` 0

- #define TRUE 1
- #define HISLIP 4880

Functions

- [int16_t start_cli](#) (void)

Variables

- [uint8_t debug_mode](#)
- [uint8_t test_mode](#)

5.8.1 Define Documentation

5.8.1.1 #define FALSE 0

5.8.1.2 #define HISLIP 4880

5.8.1.3 #define MAX_BUF_SIZE 20

5.8.1.4 #define MAX_FS 1000

5.8.1.5 #define MAX_STR_LEN 200

5.8.1.6 #define MHZ 1000000

5.8.1.7 #define TRUE 1

5.8.2 Function Documentation

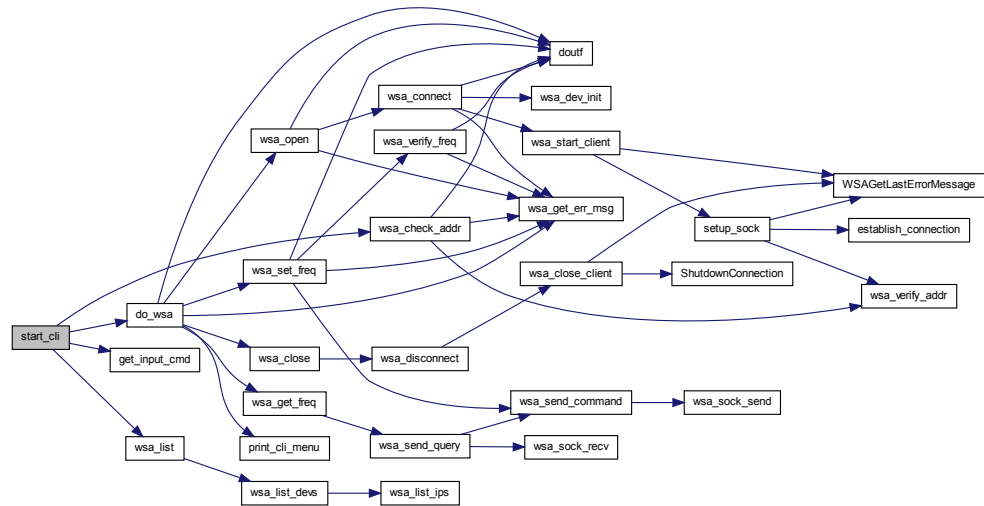
5.8.2.1 [int16_t start_cli](#) (void)

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:



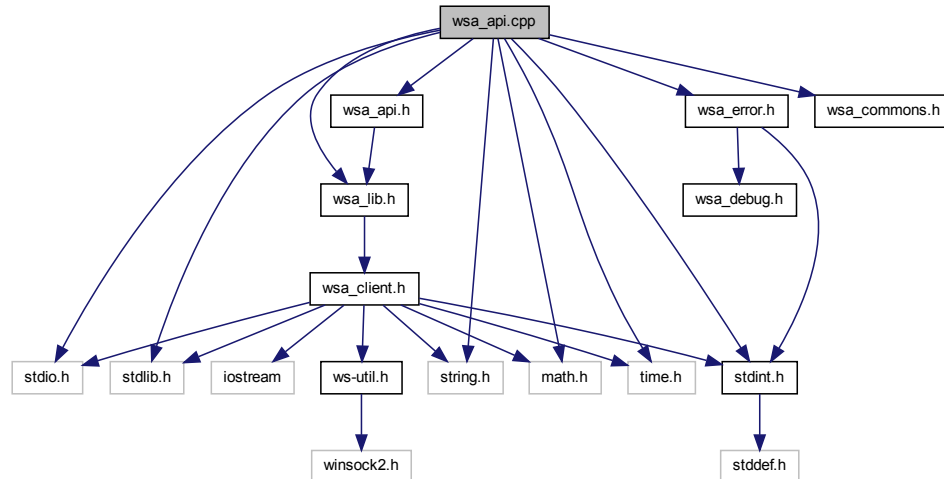
5.8.3 Variable Documentation

5.8.3.1 uint8_t debug_mode

5.8.3.2 uint8_t test_mode

5.9 wsa_api.cpp File Reference

Include dependency graph for wsa_api.cpp:



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)
- `int64_t wsa_get_freq` (struct `wsa_device` *dev)
- `int16_t wsa_set_freq` (struct `wsa_device` *dev, `uint64_t` cfreq)
- `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)
- `wsa_gain wsa_get_gain` (struct `wsa_device` *dev)
- `int16_t wsa_set_gain` (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` option)
- `int16_t wsa_check_cal_mode` (struct `wsa_device` *dev)
- `int16_t wsa_run_cal_mode` (struct `wsa_device` *dev, `uint8_t` mode)

5.9.1 Function Documentation

5.9.1.1 `int16_t wsa_check_addr (char * ip_addr)`

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

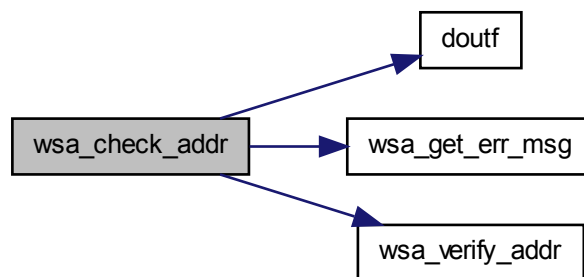
Parameters

<i>ip_addr</i> - A char pointer to the IP address or host name to be verified.
--

Returns

1 if the IP is valid, 0 if invalid (?), or a negative number on error.

Here is the call graph for this function:



5.9.1.2 `int16_t wsa_check_cal_mode (struct wsa_device * dev)`

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

Parameters

<i>dev</i> - 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.

5.9.1.3 `void wsa_close (struct wsa_device * dev)`

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

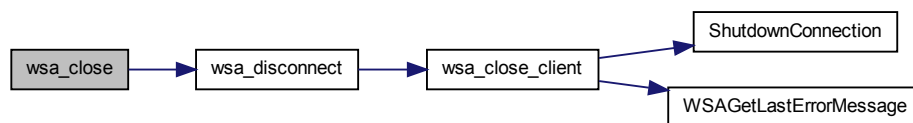
Parameters

<i>dev</i>	- A pointer to a WSA device structure to be closed.
------------	---

Returns

none

Here is the call graph for this function:



5.9.1.4 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 unit at the absolute maximum may cause damage to the device.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>gain</i>	- 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.

5.9.1.5 int16_t wsa_get_antenna (struct wsa_device * dev)

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

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

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

5.9.1.6 int16_t wsa_get_bpf (struct wsa_device * dev)

Gets the current mode of the RFE's internal BPF.

Parameters

dev - A pointer to the WSA device structure.

Returns

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

5.9.1.7 int64_t wsa_get_freq (struct wsa_device * dev)

Retrieves the center frequency that the WSA is running at.

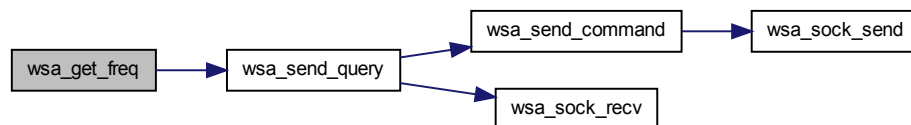
Parameters

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:

**5.9.1.8 wsa_gain wsa_get_gain (struct wsa_device * dev)**

Gets the current gain setting of the WSA.

Parameters

dev - A pointer to the WSA device structure.

Returns

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

5.9.1.9 int16_t wsa_get_lpf (struct wsa_device * dev)

Gets the current mode of the RFE's internal LPF.

Parameters

dev - A pointer to the WSA device structure.

Returns

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

5.9.1.10 int16_t wsa_is_connected (struct wsa_device * dev)

Indicates if the WSA is still connected to the PC.

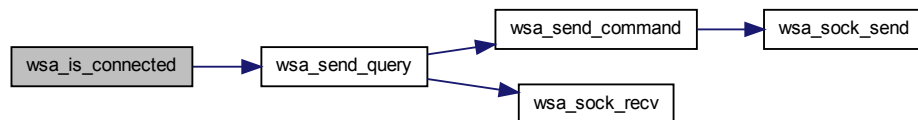
Parameters

<i>dev</i> - 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:

**5.9.1.11 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?

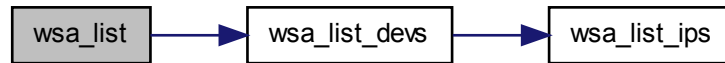
Parameters

<i>wsa_list</i> - 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:



5.9.1.12 `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

<i>dev</i>	- A pointer to the WSA device structure to be opened.
<i>intf_method</i>	- A char pointer to store the interface method to the WSA. Possible methods: <ul style="list-style-type: none">• With LAN, use: "TCPIP::<ip address="" of="" the="" wsa="">::HISLIP"</ip>• With USB, use: "USB" (check if supported with the WSA version used).

Returns

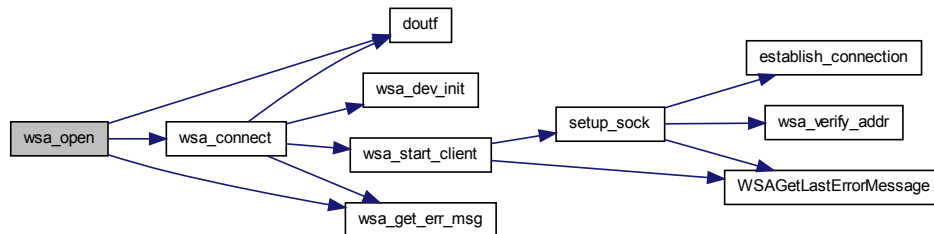
0 on success, or a negative number on error.

Errors:

Situations that will generate an error are:

- the selected connection type 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:



5.9.1.13 `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

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to wsa_frame_header structure to store information for the frame.
<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <i>sample_size</i> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the <i>sample_size</i> .
<i>sample_size</i>	- 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.

5.9.1.14 `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

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

Returns

0 on success, or a negative number on error.

5.9.1.15 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

<i>dev</i>	- A pointer to the WSA device structure.
<i>port_num</i>	- An integer port number to used. Available ports: 1, 2, 3. Note: When calibration mode is enabled through wsa_run_cal_mode() , these antenna ports will not be available. The selected port will resume when the calibration mode is set to off.

Returns

0 on success, or a negative number on error.

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

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

Parameters

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

Returns

0 on success, or a negative number on error.

5.9.1.17 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. Use [wsa_set_run_mode\(\)](#) with FREERUN to change.

Parameters

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

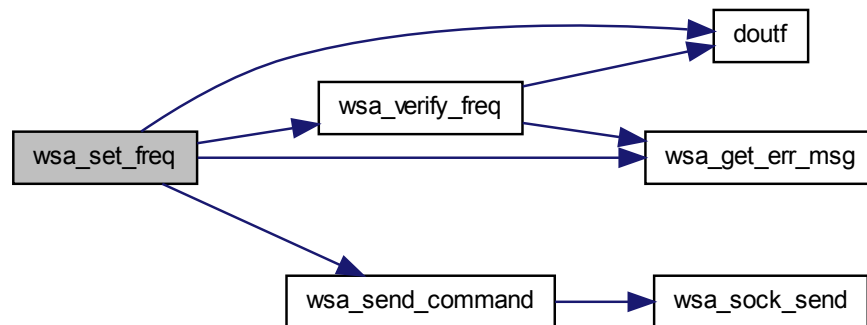
Returns

0 on success, or a negative number on error.

Errors:

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

Here is the call graph for this function:



5.9.1.18 `int16_t wsa_set_gain (struct wsa_device * dev, wsa_gain gain)`

Sets the **gain** (sensitivity) level for the radio front end of the WSA.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>gain</i>	- The gain setting of type <code>wsa_gain</code> to set for WSA. Valid gain settings are HIGH, MEDIUM, LOW, ULOW.

Returns

0 on success, or a negative number on error.

5.9.1.19 `int16_t wsa_set_lpf (struct wsa_device * dev, uint8_t option)`

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

Parameters

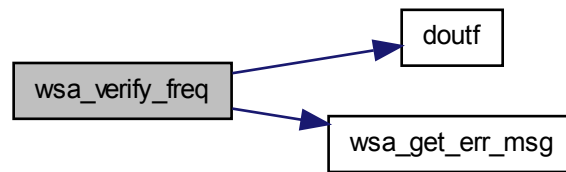
<i>dev</i>	- A pointer to the WSA device structure.
<i>option</i>	- An integer mode of selection: 0 - Off, 1 - On.

Returns

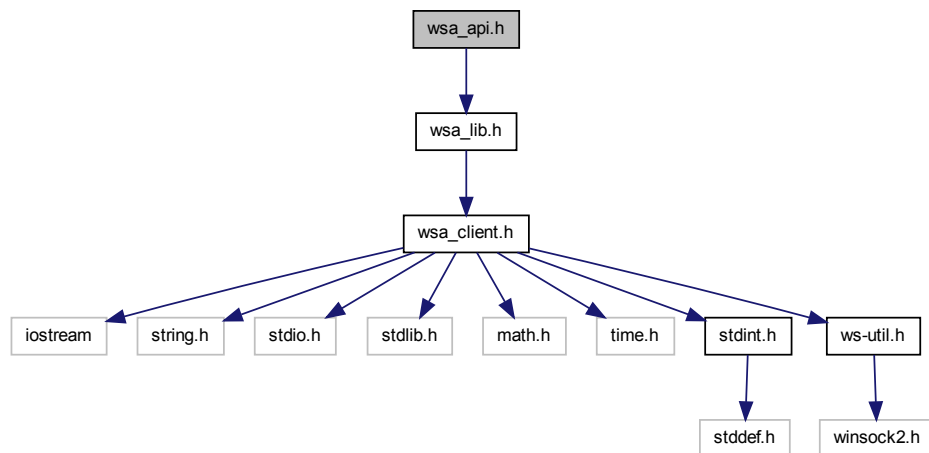
0 on success, or a negative number on error.

5.9.1.20 `int16_t wsa_verify_freq (struct wsa_device * dev, uint64_t freq)`

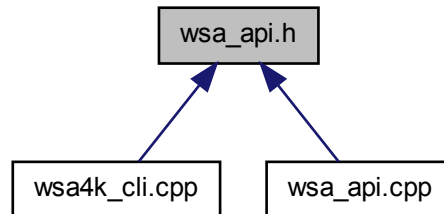
Here is the call graph for this function:

**5.10 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](#) {
 [HIGH](#) = 1, [MEDIUM](#), [LOW](#), [ULOW](#),
 [HIGH](#) = 1, [MEDIUM](#), [LOW](#), [ULOW](#) }

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)
- `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)
- `wsa_gain wsa_get_gain` (struct `wsa_device` *dev)
- `int16_t wsa_set_gain` (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` option)
- `int16_t wsa_check_cal_mode` (struct `wsa_device` *dev)
- `int16_t wsa_run_cal_mode` (struct `wsa_device` *dev, `uint8_t` mode)

5.10.1 Enumeration Type Documentation

5.10.1.1 enum wsa_gain

Defines the amplification available in the radio front end (RFE) of the WSA.

If an incorrect setting is specified, an error will be returned

Enumerator:

HIGH High RFE amplification. Value 1.

MEDIUM Medium RFE amplification.

LOW Low RFE amplification.

ULOW Ultralow RFE amplification.

HIGH

MEDIUM

LOW

ULOW

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.

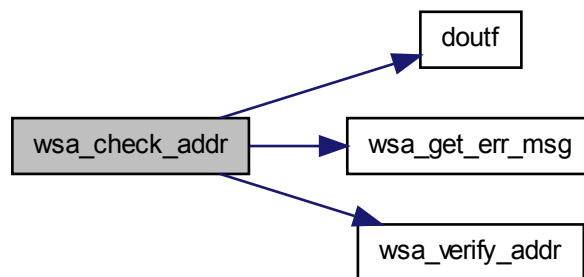
Parameters

<code>ip_addr</code> - A char pointer to the IP address or host name to be verified.
--

Returns

1 if the IP is valid, 0 if invalid (?), or a negative number on error.

Here is the call graph for this function:



5.10.2.2 int16_t wsa_check_cal_mode (struct wsa_device * dev)

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

Parameters

<i>dev</i> - 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.

5.10.2.3 void wsa_close (struct wsa_device * dev)

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

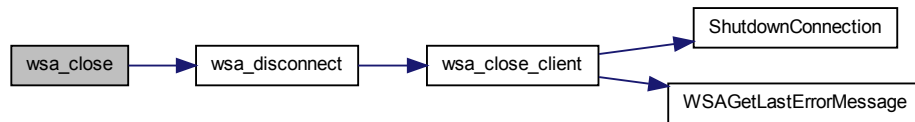
Parameters

<i>dev</i> - A pointer to a WSA device structure to be closed.
--

Returns

none

Here is the call graph for this function:



5.10.2.4 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 unit at the absolute maximum may cause damage to the device.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>gain</i>	- 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.

5.10.2.5 int16_t wsa_get_antenna (struct wsa_device * dev)

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

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

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

5.10.2.6 int16_t wsa_get_bpf (struct wsa_device * dev)

Gets the current mode of the RFE's internal BPF.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

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

5.10.2.7 `int64_t wsa_get_freq (struct wsa_device * dev)`

Retrieves the center frequency that the WSA is running at.

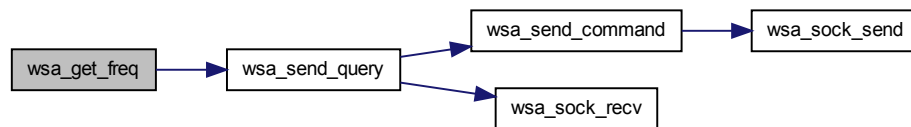
Parameters

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:



5.10.2.8 `wsa_gain_t wsa_get_gain (struct wsa_device * dev)`

Gets the current gain setting of the WSA.

Parameters

dev - A pointer to the WSA device structure.

Returns

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

5.10.2.9 `int16_t wsa_get_lpf (struct wsa_device * dev)`

Gets the current mode of the RFE's internal LPF.

Parameters

dev - A pointer to the WSA device structure.

Returns

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

5.10.2.10 `int16_t wsa_is_connected (struct wsa_device * dev)`

Indicates if the WSA is still connected to the PC.

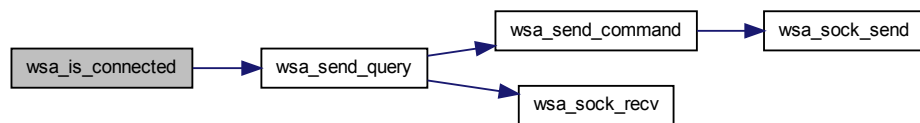
Parameters

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?

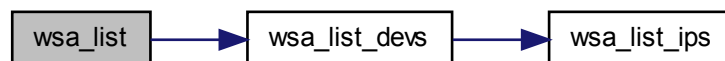
Parameters

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

<i>dev</i>	- A pointer to the WSA device structure to be opened.
<i>intf_method</i>	- A char pointer to store the interface method to the WSA. Possible methods: <ul style="list-style-type: none"> • With LAN, use: "TCP/IP::<ip address="" of="" the="" wsa="">::HISLIP"</ip> • With USB, use: "USB" (check if supported with the WSA version used).

Returns

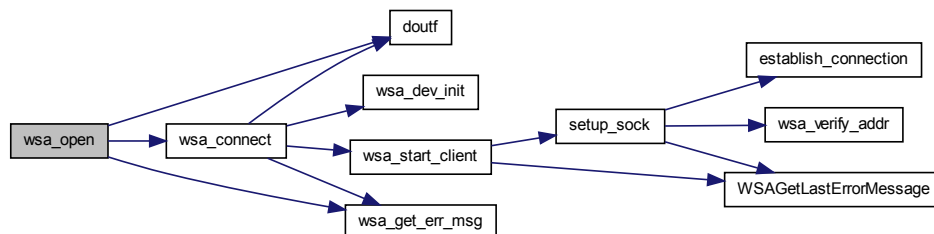
0 on success, or a negative number on error.

Errors:

Situations that will generate an error are:

- the selected connection type 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 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

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to wsa_frame_header structure to store information for the frame.

<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <code>sample_size</code> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the <code>sample_size</code> .
<i>sample_size</i>	- 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.

5.10.2.14 `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

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

Returns

0 on success, or a negative number on error.

5.10.2.15 `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

<i>dev</i>	- A pointer to the WSA device structure.
<i>port_num</i>	- An integer port number to used. Available ports: 1, 2, 3. Note: When calibration mode is enabled through wsa_run_cal_mode() , these antenna ports will not be available. The selected port will resume when the calibration mode is set to off.

Returns

0 on success, or a negative number on error.

5.10.2.16 `int16_t wsa_set_bpf (struct wsa_device * dev, uint8_t mode)`

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

Parameters

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

Returns

0 on success, or a negative number on error.

5.10.2.17 `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. Use [wsa_set_run_mode\(\)](#) with FREERUN to change.

Parameters

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

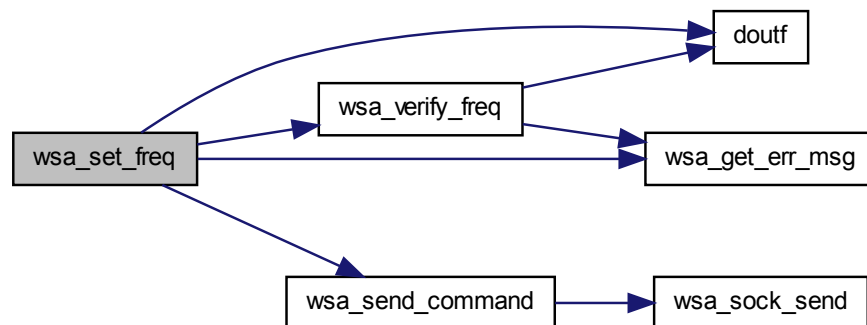
Returns

0 on success, or a negative number on error.

Errors:

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

Here is the call graph for this function:



5.10.2.18 int16_t wsa_set_gain (struct wsa_device * dev, wsa_gain gain)

Sets the **gain** (sensitivity) level for the radio front end of the WSA.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>gain</i>	- The gain setting of type wsa_gain to set for WSA. Valid gain settings are HIGH, MEDIUM, LOW, ULOW.

Returns

0 on success, or a negative number on error.

5.10.2.19 int16_t wsa_set_lpf (struct wsa_device * dev, uint8_t option)

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

Parameters

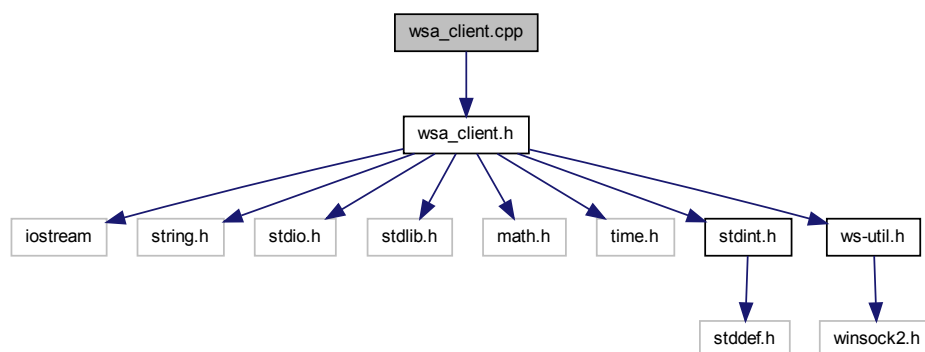
<i>dev</i>	- A pointer to the WSA device structure.
<i>option</i>	- An integer mode of selection: 0 - Off, 1 - On.

Returns

0 on success, or a negative number on error.

5.11 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](#) [test_mode](#) = FALSE
- char * [start](#) = "STARTDATA\0"
- char * [stop](#) = "STOPDATA\0"
- const [int32_t](#) [cmd_port](#) = 7000
- const [int32_t](#) [data_port](#) = 7000

5.11.1 Define Documentation

5.11.1.1 #define SHUTDOWN_DELAY

5.11.2 Function Documentation

5.11.2.1 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

<i>sock_addr</i>	-
<i>sock_port</i>	-

Returns

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

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

Here is the call graph for this function:

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

Look up, verify and establish the socket once deemed valid

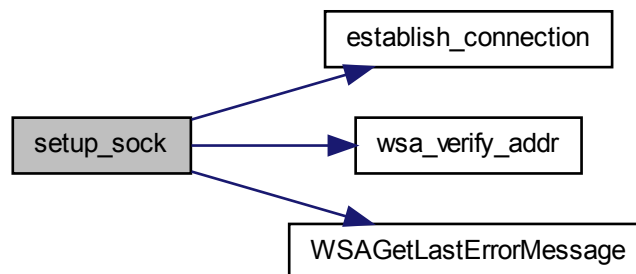
Parameters

<i>sock_name</i>	- Name of the socket (ex. server, client)
<i>sock_addr</i>	-
<i>sock_port</i>	-

Returns

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

Here is the call graph for this function:



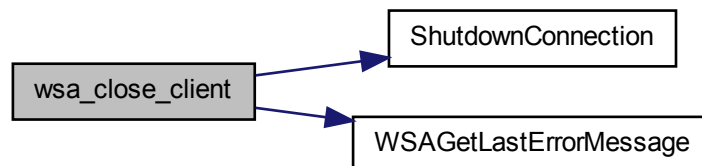
5.11.2.4 int16_t wsa_close_client (SOCKET *cmd_sock*, SOCKET *data_sock*)

Parameters

<i>cmd_sock</i>	-
<i>data_sock</i>	-

Returns

Here is the call graph for this function:



5.11.2.5 int16_t wsa_get_host_info (char * *name*)

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

Parameters

<i>name</i>	-
-------------	---

Returns

5.11.2.6 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

<i>ip_list</i>	-
----------------	---

Returns

Number of IP addresses available.

5.11.2.7 `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

<i>in_sock</i>	-
<i>rx_buf_ptr</i>	-
<i>time_out</i>	- Time out in milliseconds

Returns

Number of "words" read

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

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

Sends a string to the server.

Parameters

<i>out_sock</i>	-
<i>out_str</i>	-
<i>len</i>	-

Returns

Number of bytes sent on success, or negative otherwise.

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

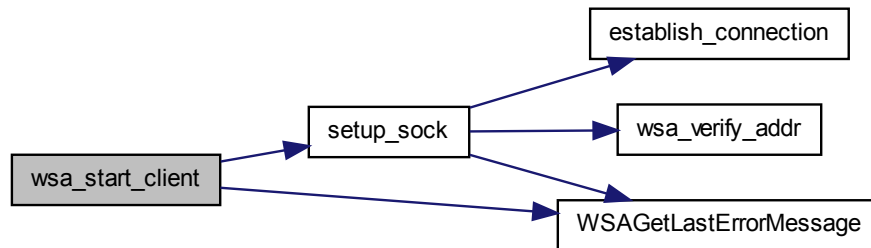
Call functions to initialize the sockets

Parameters

<i>wsa_addr</i>	-
<i>cmd_sock</i>	-
<i>data_sock</i>	-

Returns

Here is the call graph for this function:



5.11.2.11 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

<i>sock_addr</i> -

Returns

Resolved IP address or `INADDR_NONE` when failed.

5.11.3 Variable Documentation

5.11.3.1 `const int32_t cmd_port = 7000`

5.11.3.2 `const int32_t data_port = 7000`

5.11.3.3 `uint8_t debug_mode = FALSE`

5.11.3.4 `const int8_t kShutdownDelay = 1`

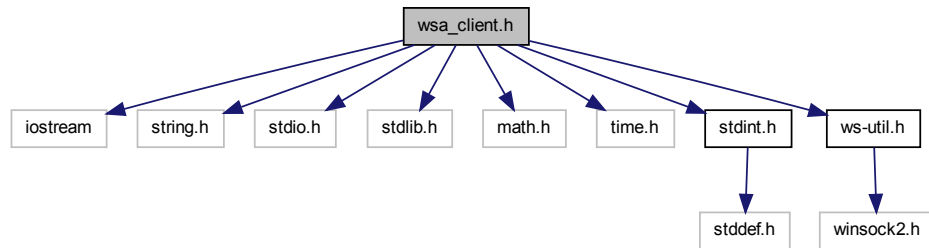
5.11.3.5 `char* start = "STARTDATA\0"`

5.11.3.6 `char* stop = "STOPDATA\0"`

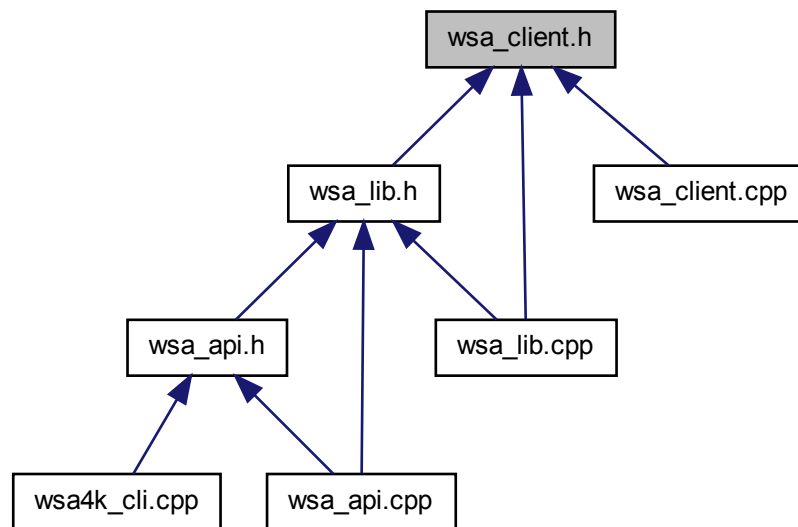
5.11.3.7 `uint8_t test_mode = FALSE`

5.12 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)

5.12.1 Define Documentation

5.12.1.1 #define HISLIP 4880

5.12.1.2 #define MAX_BUF_SIZE 20

5.12.1.3 #define MAX_STR_LEN 200

5.12.1.4 #define TIMEOUT 1000

5.12.2 Function Documentation

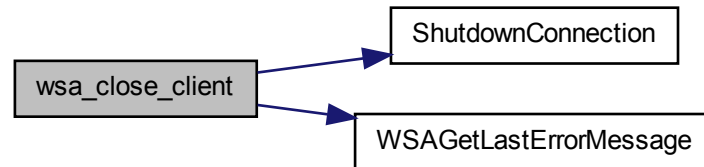
5.12.2.1 [int16_t wsa_close_client](#) (SOCKET *cmd_sock*, SOCKET *data_sock*)

Parameters

<i>cmd_sock</i>	-
<i>data_sock</i>	-

Returns

Here is the call graph for this function:



5.12.2.2 int16_t wsa_get_host_info (char * name)

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

Parameters

<i>name</i>	-
-------------	---

Returns

5.12.2.3 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

<i>ip_list</i>	-
----------------	---

Returns

Number of IP addresses available.

5.12.2.4 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

<i>in_sock</i>	-
<i>rx_buf_ptr</i>	-
<i>time_out</i>	- Time out in milliseconds

Returns

Number of "words" read

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

Sends a string to the server.

Parameters

<i>out_sock</i>	-
<i>out_str</i>	-
<i>len</i>	-

Returns

Number of bytes sent on success, or negative otherwise.

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

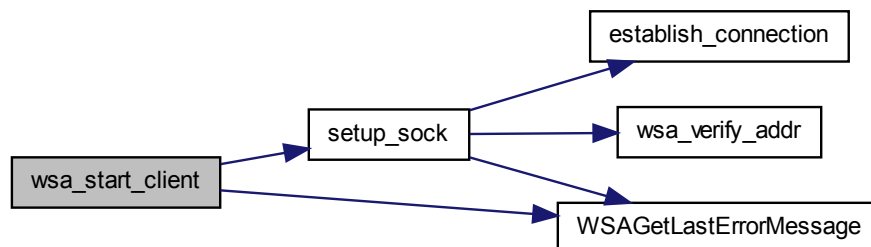
Call functions to initialize the sockets

Parameters

<i>wsa_addr</i>	-
<i>cmd_sock</i>	-
<i>data_sock</i>	-

Returns

Here is the call graph for this function:



5.12.2.7 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

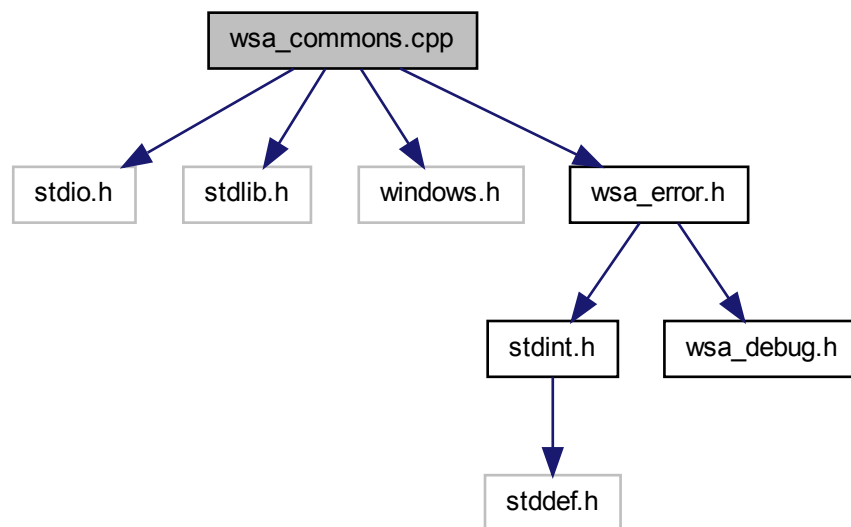
<i>sock_addr</i> -

Returns

Resolved IP address or INADDR_NONE when failed.

5.13 wsa_commons.cpp File Reference

Include dependency graph for wsa_commons.cpp:



Functions

- const char * [wsa_get_err_msg](#) (int16_t err_id)

5.13.1 Function Documentation

5.13.1.1 `const char* wsa_get_err_msg(int16_t err_id)`

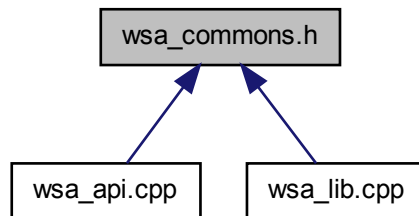
Returns the error message based on the error ID given

Parameters

<code>err_id</code> The error ID

5.14 wsa_commons.h File Reference

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



Defines

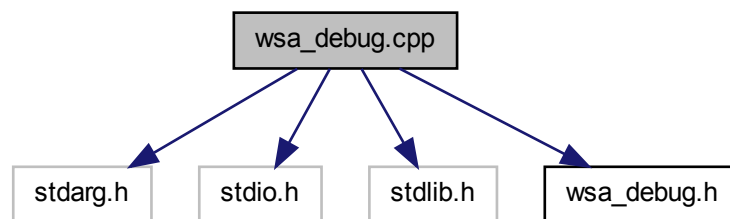
- `#define FALSE 0`
- `#define TRUE 1`
- `#define MHZ 1000000`
- `#define WSA4000 "WSA4000"`
- `#define WSA4000_INST_BW 125000000`
- `#define WSA4000_MAX_PKT_SIZE 32000`
- `#define WSA_RFE0560 "RFE0560"`
- `#define WSA_RFE0560_MAX_FREQ 7000000000`
- `#define WSA_RFE0560_MIN_FREQ 0`
- `#define WSA_RFE0560_FREQRES 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_ULOW 20`

5.14.1 Define Documentation

5.14.1.1 `#define FALSE 0`5.14.1.2 `#define MHZ 1000000`5.14.1.3 `#define TRUE 1`5.14.1.4 `#define WSA4000 "WSA4000"`5.14.1.5 `#define WSA4000_INST_BW 125000000`5.14.1.6 `#define WSA4000_MAX_PKT_SIZE 32000`5.14.1.7 `#define WSA_RFE0560 "RFE0560"`5.14.1.8 `#define WSA_RFE0560_ABS_AMP_HIGH -15`5.14.1.9 `#define WSA_RFE0560_ABS_AMP_LOW 13`5.14.1.10 `#define WSA_RFE0560_ABS_AMP_MEDIUM 0`5.14.1.11 `#define WSA_RFE0560_ABS_AMP_ULOW 20`5.14.1.12 `#define WSA_RFE0560_FREQRES 10000`5.14.1.13 `#define WSA_RFE0560_MAX_FREQ 7000000000`5.14.1.14 `#define WSA_RFE0560_MIN_FREQ 0`

5.15 wsa_debug.cpp File Reference

Include dependency graph for wsa_debug.cpp:



Functions

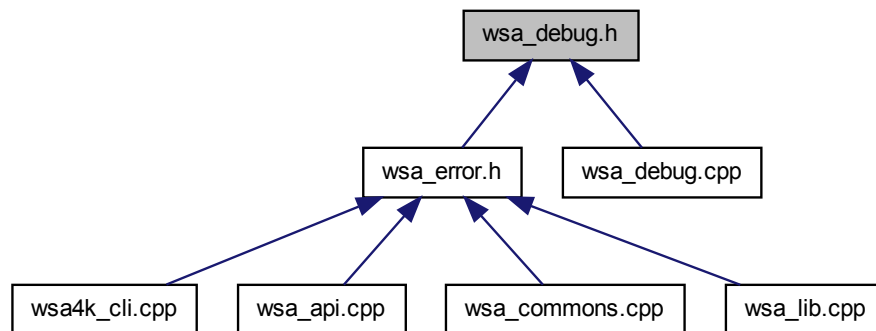
- int [doutf](#) (int level, const char *fmt,...)

5.15.1 Function Documentation

5.15.1.1 int doutf (int level, const char * fmt, ...)

5.16 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 *,...)

5.16.1 Define Documentation

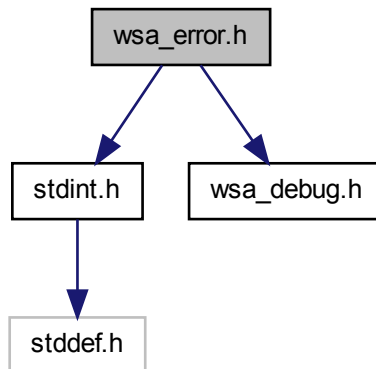
5.16.1.1 #define DEBUGLEVEL 0

5.16.2 Function Documentation

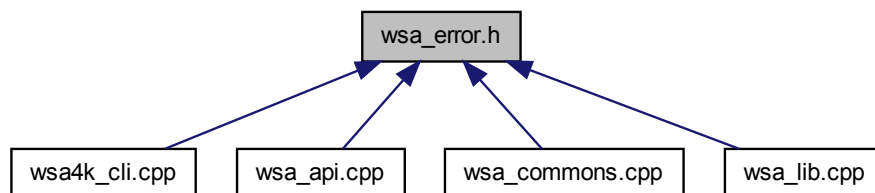
5.16.2.1 int doutf (int , const char * , ...)

5.17 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_UNKNOWNPRODSE](#) (LNEG_NUM - 4)
- #define [WSA_ERR_UNKNOWNPRODSN](#) (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_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_INVGAIN](#) (LNEG_NUM - 801)
- #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_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)`

5.17.1 Define Documentation

5.17.1.1 `#define LNEG_NUM (-10000)`

5.17.1.2 `#define WSA_ERR_CMDSENDFAILED (LNEG_NUM - 1501)`

5.17.1.3 `#define WSA_ERR_ETHERNETCONNECTFAILED (LNEG_NUM - 207)`

5.17.1.4 `#define WSA_ERR_ETHERNETINITFAILED (LNEG_NUM - 209)`

5.17.1.5 `#define WSA_ERR_ETHERNETNOTAVBL (LNEG_NUM - 206)`

5.17.1.6 `#define WSA_ERR_FREQOUTOFBOUND (LNEG_NUM - 601)`

5.17.1.7 `#define WSA_ERR_FREQSETFAILED (LNEG_NUM - 603)`

5.17.1.8 `#define WSA_ERR_INITFAILED (LNEG_NUM - 105)`

5.17.1.9 `#define WSA_ERR_INVADCCORRVALUE (LNEG_NUM - 106)`

5.17.1.10 `#define WSA_ERR_INVAMP (LNEG_NUM - 301)`

5.17.1.11 `#define WSA_ERR_INVDWELL (LNEG_NUM - 1208)`

5.17.1.12 `#define WSA_ERR_INVFREQRES (LNEG_NUM - 602)`

5.17.1.13 `#define WSA_ERR_INVGAIN (LNEG_NUM - 801)`

5.17.1.14 `#define WSA_ERR_INVINTFMETHOD (LNEG_NUM - 201)`

5.17.1.15 `#define WSA_ERR_INVIPADDRESS (LNEG_NUM - 2)`

5.17.1.16 `#define WSA_ERR_INVIPHOSTADDRESS (LNEG_NUM - 202)`

5.17.1.17 `#define WSA_ERR_INVNUMBER (LNEG_NUM - 2000)`

5.17.1.18 `#define WSA_ERR_INVNUMFRAMES (LNEG_NUM - 1209)`

5.17.1.19 `#define WSA_ERR_INVREGADDR (LNEG_NUM - 2001)`

5.17.1.20 `#define WSA_ERR_INVRUNMODE (LNEG_NUM - 1001)`

5.17.1.21 `#define WSA_ERR_INVSAMPLESIZE (LNEG_NUM - 403)`

5.17.1.22 `#define WSA_ERR_INVSTARTRES (LNEG_NUM - 1205)`

5.17.1.23 `#define WSA_ERR_INVSTOPFREQ (LNEG_NUM - 1202)`

5.17.1.24 `#define WSA_ERR_INVSTOPRES (LNEG_NUM - 1206)`

5.17.1.25 `#define WSA_ERR_INVTRIGID (LNEG_NUM - 1201)`

5.17.1.26 `#define WSA_ERR_INVTRIGRANGE (LNEG_NUM - 1207)`

5.17.1.27 `#define WSA_ERR_MALLOCF FAILED (LNEG_NUM - 2002)`

5.17.1.28 `#define WSA_ERR_NOCTRLPIPE (LNEG_NUM - 3)`

5.17.1.29 `#define WSA_ERR_NODATABUS (LNEG_NUM - 401)`

5.17.1.30 `#define WSA_ERR_NOWSA (LNEG_NUM - 1)`

5.17.1.31 `#define WSA_ERR_OPENFAILED (LNEG_NUM - 104)`

5.17.1.32 `#define WSA_ERR_PLLOCKFAILED (LNEG_NUM - 604)`

5.17.1.33 `#define WSA_ERR_PRODOBSOLETE (LNEG_NUM - 8)`

5.17.1.34 `#define WSA_ERR_READFRAMEFAILED (LNEG_NUM - 402)`

5.17.1.35 `#define WSA_ERR_SETFAILED (LNEG_NUM - 103)`

5.17.1.36 `#define WSA_ERR_STARTOOB (LNEG_NUM - 1203)`

5.17.1.37 `#define WSA_ERR_STOPOOB (LNEG_NUM - 1204)`

5.17.1.38 `#define WSA_ERR_UNKNOWN_ERROR (LNEG_NUM - 2003)`

5.17.1.39 `#define WSA_ERR_UNKNOWNFWRVSN (LNEG_NUM - 6)`

5.17.1.40 `#define WSA_ERR_UNKNOWNPRODSE (LNEG_NUM - 4)`

5.17.1.41 `#define WSA_ERR_UNKNOWNPRODVSN (LNEG_NUM - 5)`

5.17.1.42 `#define WSA_ERR_UNKNOWNRFEVSN (LNEG_NUM - 7)`

5.17.1.43 `#define WSA_ERR_USBINITFAILED (LNEG_NUM - 205)`

5.17.1.44 `#define WSA_ERR_USBNOTAVBL (LNEG_NUM - 203)`

5.17.1.45 `#define WSA_ERR_USBOPENFAILED (LNEG_NUM - 204)`

5.17.1.46 `#define WSA_ERR_WSAINUSE (LNEG_NUM - 102)`

5.17.1.47 `#define WSA_ERR_WSANOTRDY (LNEG_NUM - 101)`

5.17.2 Function Documentation

5.17.2.1 `const char* wsa_get_err_msg (int16_t err_id)`

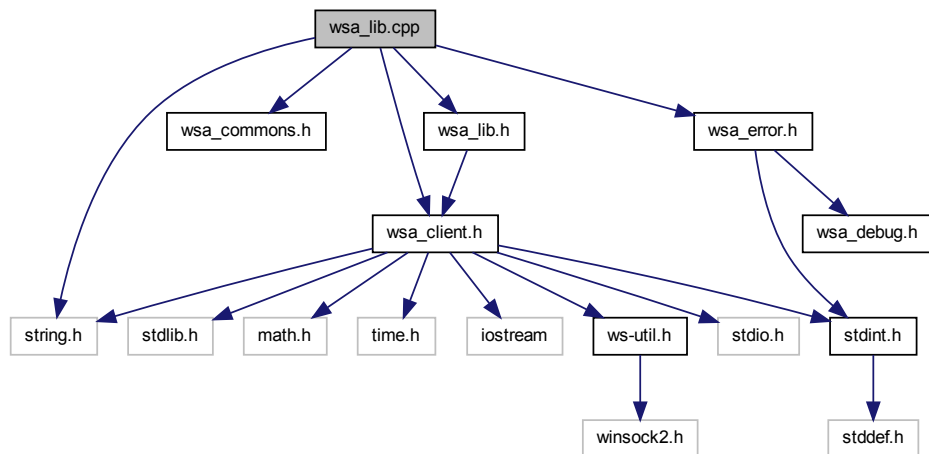
Returns the error message based on the error ID given

Parameters

<i>err_id</i> The error ID

5.18 wsa_lib.cpp File Reference

Include dependency graph for wsa_lib.cpp:



Functions

- `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)`
- `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)`

5.18.1 Function Documentation

5.18.1.1 `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.

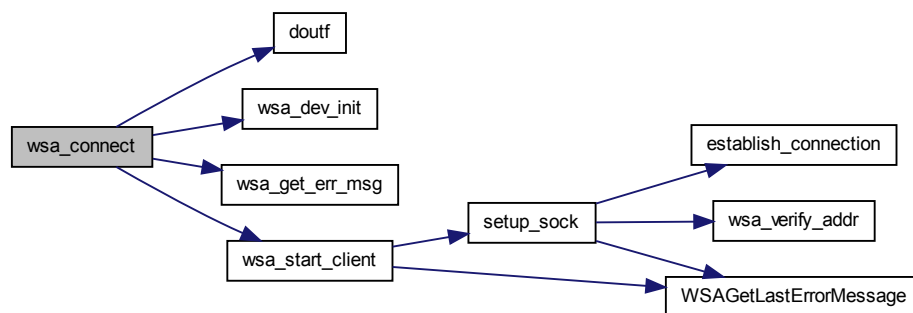
Parameters

<i>dev</i>	- A pointer to the WSA device structure to be connected/established.
<i>cmd_syntax</i>	- A char pointer to store standard for control commands communication to the WSA. Currently supported standard command syntax type is: SCPI.
<i>intf_method</i>	- A char pointer to store the interface method to the WSA. Possible methods: <ul style="list-style-type: none"> • With LAN, use: "TCP/IP::<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:



5.18.1.2 int16_t wsa_dev_init (struct wsa_device * dev)

Initialized the the [wsa_device](#) structure

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

None

5.18.1.3 `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.

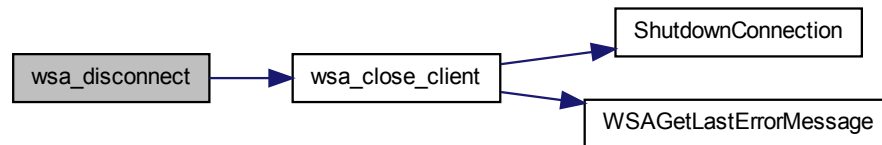
Parameters

<i>dev</i> - 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:



5.18.1.4 `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

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to wsa_frame_header structure to store information for the frame.
<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <code>sample_size</code> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the <code>sample_size</code> .
<i>sample_size</i>	- 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.

5.18.1.5 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

<i>wsa_list</i>	- 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:



5.18.1.6 int16_t wsa_query_error (struct wsa_device * dev)

Query the WSA for any error.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

0 on success, or a negative number on error.

5.18.1.7 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

<i>dev</i>	- 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\(\)](#).

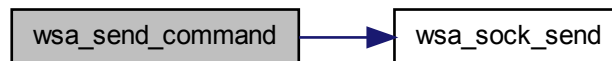
Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>command</i>	- 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:



5.18.1.8 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\(\)](#).

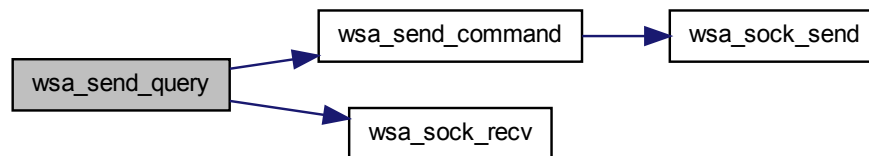
Parameters

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

Returns

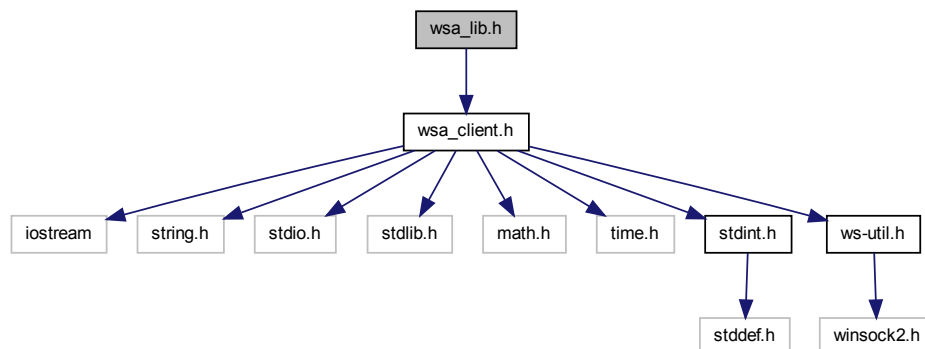
The result stored in a [wsa_resp](#) struct format.

Here is the call graph for this function:

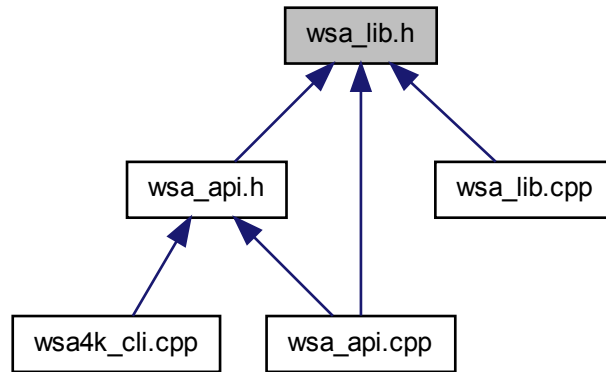


5.19 wsa_lib.h File Reference

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 [SCPI](#) "SCPI"

Enumerations

- enum [wsa_gain](#) {
[HIGH](#) = 1, [MEDIUM](#), [LOW](#), [ULOW](#),
[HIGH](#) = 1, [MEDIUM](#), [LOW](#), [ULOW](#) }

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)
- 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 FALSE 0

5.19.1.2 #define SCPI "SCPI"

5.19.1.3 #define TRUE 1

5.19.2 Enumeration Type Documentation

5.19.2.1 enum [wsa_gain](#)

Enumerator:

HIGH High RFE amplification. Value 1.

MEDIUM Medium RFE amplification.

LOW Low RFE amplification.

ULOW Ultralow RFE amplification.

HIGH

MEDIUM

LOW

ULOW

5.19.3 Function Documentation

5.19.3.1 `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.

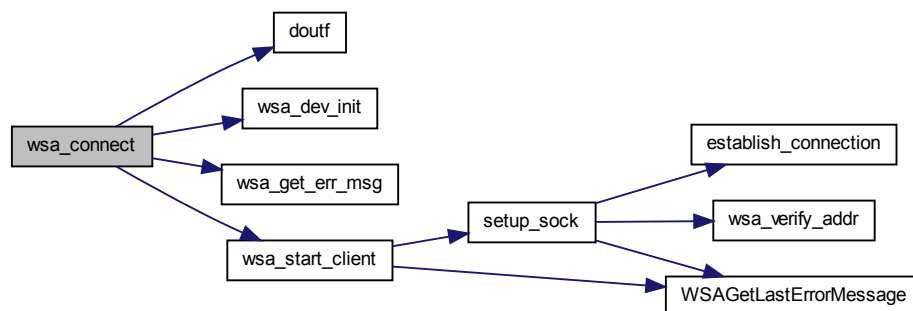
Parameters

<i>dev</i>	- A pointer to the WSA device structure to be connected/established.
<i>cmd_syntax</i>	- A char pointer to store standard for control commands communication to the WSA. Currently supported standard command syntax type is: SCPI.
<i>intf_method</i>	- A char pointer to store the interface method to the WSA. Possible methods: <ul style="list-style-type: none"> • With LAN, use: "TCPIP::<ip address="" of="" the="" wsa="">::HISLIP"</ip> • 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:



5.19.3.2 `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.

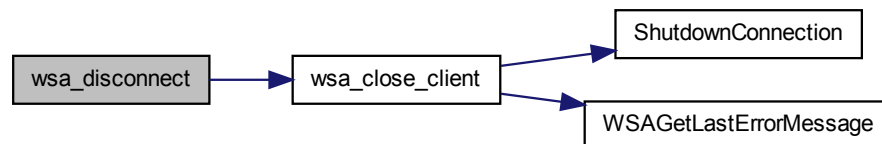
Parameters

<i>dev</i>	- 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:



5.19.3.3 `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

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to wsa_frame_header structure to store information for the frame.
<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <code>sample_size</code> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled Q data buffer with size specified by the <code>sample_size</code> .
<i>sample_size</i>	- 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.

5.19.3.4 `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

<i>wsa_list</i>	- 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:



5.19.3.5 `int16_t wsa_query_error (struct wsa_device * dev)`

Query the WSA for any error.

Parameters

<i>dev</i>	- A pointer to the WSA device structure.
------------	--

Returns

0 on success, or a negative number on error.

5.19.3.6 `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

<i>dev</i>	- 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

<i>dev</i>	- A pointer to the WSA device structure.
<i>command</i>	- 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:



5.19.3.7 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\(\)](#).

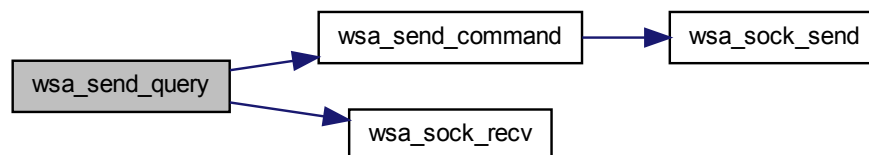
Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>command</i>	- 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:



5.20 wsa_lib.txt File Reference

Contain some code documents for [wsa_lib.h](#).

5.20.1 Detailed Description

Index

_WIN32_WINNT
targetver.h, 11

cmd
wsa_socket, 7
cmd_port
wsa_client.cpp, 45

configurations
ReadMe.txt, 9

data
wsa_socket, 7
data_port
wsa_client.cpp, 45

debug_mode
wsa4k_cli.h, 20
wsa_client.cpp, 45

DEBUGLEVEL
wsa_debug.h, 53

descr
wsa_device, 5

do_wsa
wsa4k_cli.cpp, 15

doutf
wsa_debug.cpp, 53
wsa_debug.h, 53

establish_connection
wsa_client.cpp, 41

FALSE
wsa4k_cli.h, 19
wsa_commons.h, 52
wsa_lib.h, 65

files
ReadMe.txt, 9

freq
wsa_frame_header, 6

fw_version
wsa_descriptor, 4

gain
wsa_frame_header, 6

get_input_cmd
wsa4k_cli.cpp, 16

get_sock_ack
wsa_client.cpp, 42

HIGH
wsa_api.h, 32
wsa_lib.h, 65

HISLIP
wsa4k_cli.h, 19
wsa_client.h, 47

inst_bw
wsa_descriptor, 4

int16_t
stdint.h, 11

int32_t
stdint.h, 11

int64_t
stdint.h, 11

int8_t
stdint.h, 11

intf_type
wsa_descriptor, 4

kBufferSize
ws-util.cpp, 12

kNumMessages
ws-util.cpp, 13

kShutdownDelay
wsa_client.cpp, 45

LNEG_NUM
wsa_error.h, 56

LOW
wsa_api.h, 32
wsa_lib.h, 65

main
main.cpp, 9

main.cpp, 8
main, 9

MAX_BUF_SIZE
wsa4k_cli.h, 19
wsa_client.h, 47

MAX_FS
wsa4k_cli.h, 19

max_sample_size
wsa_descriptor, 4

MAX_STR_LEN
wsa4k_cli.h, 19
wsa_client.h, 47

- max_tune_freq
 - wsa_descriptor, [4](#)
- MEDIUM
 - wsa_api.h, [32](#)
 - wsa_lib.h, [65](#)
- MHZ
 - wsa4k_cli.h, [19](#)
 - wsa_commons.h, [52](#)
- min_tune_freq
 - wsa_descriptor, [4](#)
- nsec
 - wsa_time, [8](#)
- platforms
 - ReadMe.txt, [9](#)
- print_cli_menu
 - wsa4k_cli.cpp, [16](#)
- prod_name
 - wsa_descriptor, [4](#)
- prod_serial
 - wsa_descriptor, [4](#)
 - wsa_frame_header, [6](#)
- prod_version
 - wsa_descriptor, [4](#)
- ReadMe.txt, [9](#)
 - configurations, [9](#)
 - files, [9](#)
 - platforms, [9](#)
- result
 - wsa_resp, [7](#)
- rfe_name
 - wsa_descriptor, [4](#)
- rfe_version
 - wsa_descriptor, [4](#)
- sample_size
 - wsa_frame_header, [6](#)
- SCPI
 - wsa_lib.h, [65](#)
- sec
 - wsa_time, [8](#)
- setup_sock
 - wsa_client.cpp, [42](#)
- SHUTDOWN_DELAY
 - wsa_client.cpp, [41](#)
- ShutdownConnection
 - ws-util.cpp, [12](#)
 - ws-util.h, [14](#)
- sock
 - wsa_device, [5](#)
- start
 - wsa_client.cpp, [45](#)
- start_cli
 - wsa4k_cli.cpp, [16](#)
 - wsa4k_cli.h, [19](#)
- status
 - wsa_resp, [7](#)
- stdint.h, [10](#)
 - int16_t, [11](#)
 - int32_t, [11](#)
 - int64_t, [11](#)
 - int8_t, [11](#)
 - uint16_t, [11](#)
 - uint32_t, [11](#)
 - uint64_t, [11](#)
 - uint8_t, [11](#)
- stop
 - wsa_client.cpp, [45](#)
- targetver.h, [11](#)
 - _WIN32_WINNT, [11](#)
- test_mode
 - wsa4k_cli.h, [20](#)
 - wsa_client.cpp, [45](#)
- time_stamp
 - wsa_frame_header, [6](#)
- TIMEOUT
 - wsa_client.h, [47](#)
- TRUE
 - wsa4k_cli.h, [19](#)
 - wsa_commons.h, [52](#)
 - wsa_lib.h, [65](#)
- uint16_t
 - stdint.h, [11](#)
- uint32_t
 - stdint.h, [11](#)
- uint64_t
 - stdint.h, [11](#)
- uint8_t
 - stdint.h, [11](#)
- ULOW
 - wsa_api.h, [32](#)
 - wsa_lib.h, [65](#)
- ws-util.cpp, [12](#)
 - kBufferSize, [12](#)
 - kNumMessages, [13](#)

- ShutdownConnection, [12](#)
- WSAGetLastError, [12](#)
- ws-util.h, [13](#)
 - ShutdownConnection, [14](#)
 - WSAGetLastError, [14](#)
- WSA4000
 - wsa_commons.h, [52](#)
- WSA4000_INST_BW
 - wsa_commons.h, [52](#)
- WSA4000_MAX_PKT_SIZE
 - wsa_commons.h, [52](#)
- wsa4k_cli.cpp, [15](#)
 - do_wsa, [15](#)
 - get_input_cmd, [16](#)
 - print_cli_menu, [16](#)
 - start_cli, [16](#)
- wsa4k_cli.h, [18](#)
 - debug_mode, [20](#)
 - FALSE, [19](#)
 - HISLIP, [19](#)
 - MAX_BUF_SIZE, [19](#)
 - MAX_FS, [19](#)
 - MAX_STR_LEN, [19](#)
 - MHZ, [19](#)
 - start_cli, [19](#)
 - test_mode, [20](#)
 - TRUE, [19](#)
- wsa_api.h
 - HIGH, [32](#)
 - LOW, [32](#)
 - MEDIUM, [32](#)
 - ULOW, [32](#)
- wsa_lib.h
 - HIGH, [65](#)
 - LOW, [65](#)
 - MEDIUM, [65](#)
 - ULOW, [65](#)
- wsa_api.cpp, [21](#)
 - wsa_check_addr, [22](#)
 - wsa_check_cal_mode, [22](#)
 - wsa_close, [22](#)
 - wsa_get_abs_max_amp, [23](#)
 - wsa_get_antenna, [23](#)
 - wsa_get_bpf, [23](#)
 - wsa_get_freq, [24](#)
 - wsa_get_gain, [24](#)
 - wsa_get_lpf, [24](#)
 - wsa_is_connected, [25](#)
 - wsa_list, [25](#)
 - wsa_open, [26](#)
 - wsa_read_pkt, [27](#)
 - wsa_run_cal_mode, [27](#)
 - wsa_set_antenna, [28](#)
 - wsa_set_bpf, [28](#)
 - wsa_set_freq, [28](#)
 - wsa_set_gain, [29](#)
 - wsa_set_lpf, [29](#)
 - wsa_verify_freq, [30](#)
- wsa_api.h, [30](#)
 - wsa_check_addr, [32](#)
 - wsa_check_cal_mode, [33](#)
 - wsa_close, [33](#)
 - wsa_gain, [32](#)
 - wsa_get_abs_max_amp, [34](#)
 - wsa_get_antenna, [34](#)
 - wsa_get_bpf, [34](#)
 - wsa_get_freq, [34](#)
 - wsa_get_gain, [35](#)
 - wsa_get_lpf, [35](#)
 - wsa_is_connected, [35](#)
 - wsa_list, [36](#)
 - wsa_open, [36](#)
 - wsa_read_pkt, [37](#)
 - wsa_run_cal_mode, [38](#)
 - wsa_set_antenna, [38](#)
 - wsa_set_bpf, [38](#)
 - wsa_set_freq, [39](#)
 - wsa_set_gain, [39](#)
 - wsa_set_lpf, [40](#)
- wsa_check_addr
 - wsa_api.cpp, [22](#)
 - wsa_api.h, [32](#)
- wsa_check_cal_mode
 - wsa_api.cpp, [22](#)
 - wsa_api.h, [33](#)
- wsa_client.cpp, [40](#)
 - cmd_port, [45](#)
 - data_port, [45](#)
 - debug_mode, [45](#)
 - establish_connection, [41](#)
 - get_sock_ack, [42](#)
 - kShutdownDelay, [45](#)
 - setup_sock, [42](#)
 - SHUTDOWN_DELAY, [41](#)
 - start, [45](#)
 - stop, [45](#)
 - test_mode, [45](#)
 - wsa_close_client, [43](#)
 - wsa_get_host_info, [43](#)
 - wsa_list_ips, [43](#)

- wsa_sock_recv, [43](#)
- wsa_sock_recv_words, [44](#)
- wsa_sock_send, [44](#)
- wsa_start_client, [44](#)
- wsa_verify_addr, [45](#)
- wsa_client.h, [46](#)
- HISLIP, [47](#)
- MAX_BUF_SIZE, [47](#)
- MAX_STR_LEN, [47](#)
- TIMEOUT, [47](#)
- wsa_close_client, [47](#)
- wsa_get_host_info, [48](#)
- wsa_list_ips, [48](#)
- wsa_sock_recv, [48](#)
- wsa_sock_send, [49](#)
- wsa_start_client, [49](#)
- wsa_verify_addr, [49](#)
- wsa_close
 - wsa_api.cpp, [22](#)
 - wsa_api.h, [33](#)
- wsa_close_client
 - wsa_client.cpp, [43](#)
 - wsa_client.h, [47](#)
- wsa_commons.cpp, [50](#)
- wsa_get_err_msg, [50](#)
- wsa_commons.h, [51](#)
- FALSE, [52](#)
- MHZ, [52](#)
- TRUE, [52](#)
- WSA4000, [52](#)
- WSA4000_INST_BW, [52](#)
- WSA4000_MAX_PKT_SIZE, [52](#)
- WSA_RFE0560, [52](#)
- WSA_RFE0560_ABS_AMP_HIGH, [52](#)
- WSA_RFE0560_ABS_AMP_LOW, [52](#)
- WSA_RFE0560_ABS_AMP_MEDIUM, [52](#)
- WSA_RFE0560_ABS_AMP_ULONG, [52](#)
- WSA_RFE0560_FREQRES, [52](#)
- WSA_RFE0560_MAX_FREQ, [52](#)
- WSA_RFE0560_MIN_FREQ, [52](#)
- wsa_connect
 - wsa_lib.cpp, [58](#)
 - wsa_lib.h, [65](#)
- wsa_debug.cpp, [52](#)
- doutf, [53](#)
- wsa_debug.h, [53](#)
- DEBUGLEVEL, [53](#)
- doutf, [53](#)
- wsa_descriptor, [3](#)
- fw_version, [4](#)
- inst_bw, [4](#)
- intf_type, [4](#)
- max_sample_size, [4](#)
- max_tune_freq, [4](#)
- min_tune_freq, [4](#)
- prod_name, [4](#)
- prod_serial, [4](#)
- prod_version, [4](#)
- rfe_name, [4](#)
- rfe_version, [4](#)
- wsa_dev_init
 - wsa_lib.cpp, [59](#)
- wsa_device, [4](#)
- descr, [5](#)
- sock, [5](#)
- wsa_disconnect
 - wsa_lib.cpp, [59](#)
 - wsa_lib.h, [66](#)
- WSA_ERR_CMDSENDFAILED
 - wsa_error.h, [56](#)
- WSA_ERR_ETHERNETCONNECTFAILED
 - wsa_error.h, [56](#)
- WSA_ERR_ETHERNETINITFAILED
 - wsa_error.h, [56](#)
- WSA_ERR_ETHERNETNOTAVBL
 - wsa_error.h, [56](#)
- WSA_ERR_FREQOUTOFBOUND
 - wsa_error.h, [56](#)
- WSA_ERR_FREQSETFAILED
 - wsa_error.h, [56](#)
- WSA_ERR_INITFAILED
 - wsa_error.h, [56](#)
- WSA_ERR_INVADCCORRVALUE
 - wsa_error.h, [56](#)
- WSA_ERR_INVAMP
 - wsa_error.h, [56](#)
- WSA_ERR_INVDWELL
 - wsa_error.h, [56](#)
- WSA_ERR_INVFREQRES
 - wsa_error.h, [56](#)
- WSA_ERR_INVGAIN
 - wsa_error.h, [56](#)
- WSA_ERR_INVINTFMETHOD
 - wsa_error.h, [56](#)
- WSA_ERR_INVIPADDRESS
 - wsa_error.h, [56](#)
- WSA_ERR_INVIPHOSTADDRESS
 - wsa_error.h, [56](#)
- WSA_ERR_INVNUMBER

- [wsa_error.h](#), [56](#)
- WSA_ERR_INVNUMFRAMES
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVREGADDR
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVRUNMODE
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVSAMPLESIZE
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVSTARTRES
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVSTOPFREQ
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVSTOPRES
 - [wsa_error.h](#), [56](#)
- WSA_ERR_INVTRIGID
 - [wsa_error.h](#), [57](#)
- WSA_ERR_INVTRIGRANGE
 - [wsa_error.h](#), [57](#)
- WSA_ERR_MALLOCF FAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_NOCTRLPIPE
 - [wsa_error.h](#), [57](#)
- WSA_ERR_NODATABUS
 - [wsa_error.h](#), [57](#)
- WSA_ERR_NOWSA
 - [wsa_error.h](#), [57](#)
- WSA_ERR_OPENFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_PLLOCKFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_PRODOBSOLETE
 - [wsa_error.h](#), [57](#)
- WSA_ERR_READFRAMEFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_SETFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_STARTOOB
 - [wsa_error.h](#), [57](#)
- WSA_ERR_STOPOOB
 - [wsa_error.h](#), [57](#)
- WSA_ERR_UNKNOWN_ERROR
 - [wsa_error.h](#), [57](#)
- WSA_ERR_UNKNOWNFWRVSN
 - [wsa_error.h](#), [57](#)
- WSA_ERR_UNKNOWNPRODSEr
 - [wsa_error.h](#), [57](#)
- WSA_ERR_UNKNOWNPRODVSN
 - [wsa_error.h](#), [57](#)
- WSA_ERR_UNKNOWNRFEVSN
 - [wsa_error.h](#), [57](#)
- [wsa_error.h](#), [57](#)
- WSA_ERR_USBINITFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_USBNOTAVBL
 - [wsa_error.h](#), [57](#)
- WSA_ERR_USBOPENFAILED
 - [wsa_error.h](#), [57](#)
- WSA_ERR_WSAINUSE
 - [wsa_error.h](#), [57](#)
- WSA_ERR_WSANOTRDY
 - [wsa_error.h](#), [57](#)
- [wsa_error.h](#), [54](#)
 - [LNEG_NUM](#), [56](#)
 - [WSA_ERR_CMDS ENDF AILED](#), [56](#)
 - [WSA_ERR_ETHERNETCONNECTFAILED](#), [56](#)
 - [WSA_ERR_ETHERNETINITFAILED](#), [56](#)
 - [WSA_ERR_ETHERNETNOTAVBL](#), [56](#)
 - [WSA_ERR_FREQOUTOFBOUND](#), [56](#)
 - [WSA_ERR_FREQSETFAILED](#), [56](#)
 - [WSA_ERR_INITFAILED](#), [56](#)
 - [WSA_ERR_INVADCCORRVALUE](#), [56](#)
 - [WSA_ERR_INVAMP](#), [56](#)
 - [WSA_ERR_INVDWELL](#), [56](#)
 - [WSA_ERR_INVFREQR ES](#), [56](#)
 - [WSA_ERR_INVGAIN](#), [56](#)
 - [WSA_ERR_INVINTFMETHOD](#), [56](#)
 - [WSA_ERR_INVIPADDRESS](#), [56](#)
 - [WSA_ERR_INVIPHOSTADDRESS](#), [56](#)
 - [WSA_ERR_INVNUMBER](#), [56](#)
 - [WSA_ERR_INVNUMFRAMES](#), [56](#)
 - [WSA_ERR_INVREGADDR](#), [56](#)
 - [WSA_ERR_INVRUNMODE](#), [56](#)
 - [WSA_ERR_INVSAMPLESIZE](#), [56](#)
 - [WSA_ERR_INVSTARTRES](#), [56](#)
 - [WSA_ERR_INVSTOPFREQ](#), [56](#)
 - [WSA_ERR_INVSTOPRES](#), [56](#)
 - [WSA_ERR_INVTRIGID](#), [57](#)
 - [WSA_ERR_INVTRIGRANGE](#), [57](#)
 - [WSA_ERR_MALLOCF AILED](#), [57](#)
 - [WSA_ERR_NOCTRLPIPE](#), [57](#)
 - [WSA_ERR_NODATABUS](#), [57](#)
 - [WSA_ERR_NOWSA](#), [57](#)
 - [WSA_ERR_OPENFAILED](#), [57](#)
 - [WSA_ERR_PLLOCKFAILED](#), [57](#)
 - [WSA_ERR_PRODOBSOLETE](#), [57](#)
 - [WSA_ERR_READFRAMEFAILED](#), [57](#)
 - [WSA_ERR_SETFAILED](#), [57](#)
 - [WSA_ERR_STARTOOB](#), [57](#)

- WSA_ERR_STOPOOB, [57](#)
- WSA_ERR_UNKNOWN_ERROR, [57](#)
- WSA_ERR_UNKNOWNFWRVSN, [57](#)
- WSA_ERR_UNKNOWNPRODSE, [57](#)
- WSA_ERR_UNKNOWNPRODVS, [57](#)
- WSA_ERR_UNKNOWNRFEVSN, [57](#)
- WSA_ERR_USBINITFAILED, [57](#)
- WSA_ERR_USBNOTAVBL, [57](#)
- WSA_ERR_USBOPENFAILED, [57](#)
- WSA_ERR_WSAINUSE, [57](#)
- WSA_ERR_WSANOTRDY, [57](#)
- wsa_get_err_msg, [57](#)
- wsa_frame_header, [5](#)
 - freq, [6](#)
 - gain, [6](#)
 - prod_serial, [6](#)
 - sample_size, [6](#)
 - time_stamp, [6](#)
- wsa_gain
 - wsa_api.h, [32](#)
 - wsa_lib.h, [65](#)
- wsa_get_abs_max_amp
 - wsa_api.cpp, [23](#)
 - wsa_api.h, [34](#)
- wsa_get_antenna
 - wsa_api.cpp, [23](#)
 - wsa_api.h, [34](#)
- wsa_get_bpf
 - wsa_api.cpp, [23](#)
 - wsa_api.h, [34](#)
- wsa_get_err_msg
 - wsa_commons.cpp, [50](#)
 - wsa_error.h, [57](#)
- wsa_get_frame
 - wsa_lib.cpp, [60](#)
 - wsa_lib.h, [67](#)
- wsa_get_freq
 - wsa_api.cpp, [24](#)
 - wsa_api.h, [34](#)
- wsa_get_gain
 - wsa_api.cpp, [24](#)
 - wsa_api.h, [35](#)
- wsa_get_host_info
 - wsa_client.cpp, [43](#)
 - wsa_client.h, [48](#)
- wsa_get_lpf
 - wsa_api.cpp, [24](#)
 - wsa_api.h, [35](#)
- wsa_is_connected
 - wsa_api.cpp, [25](#)
- wsa_api.h, [35](#)
- wsa_lib.cpp, [58](#)
 - wsa_connect, [58](#)
 - wsa_dev_init, [59](#)
 - wsa_disconnect, [59](#)
 - wsa_get_frame, [60](#)
 - wsa_list_devs, [60](#)
 - wsa_query_error, [61](#)
 - wsa_send_command, [61](#)
 - wsa_send_query, [62](#)
- wsa_lib.h, [63](#)
 - FALSE, [65](#)
 - SCPI, [65](#)
 - TRUE, [65](#)
 - wsa_connect, [65](#)
 - wsa_disconnect, [66](#)
 - wsa_gain, [65](#)
 - wsa_get_frame, [67](#)
 - wsa_list_devs, [67](#)
 - wsa_query_error, [68](#)
 - wsa_send_command, [68](#)
 - wsa_send_query, [69](#)
- wsa_lib.txt, [70](#)
- wsa_list
 - wsa_api.cpp, [25](#)
 - wsa_api.h, [36](#)
- wsa_list_devs
 - wsa_lib.cpp, [60](#)
 - wsa_lib.h, [67](#)
- wsa_list_ips
 - wsa_client.cpp, [43](#)
 - wsa_client.h, [48](#)
- wsa_open
 - wsa_api.cpp, [26](#)
 - wsa_api.h, [36](#)
- wsa_query_error
 - wsa_lib.cpp, [61](#)
 - wsa_lib.h, [68](#)
- wsa_read_pkt
 - wsa_api.cpp, [27](#)
 - wsa_api.h, [37](#)
- wsa_resp, [7](#)
 - result, [7](#)
 - status, [7](#)
- WSA_RFE0560
 - wsa_commons.h, [52](#)
- WSA_RFE0560_ABS_AMP_HIGH
 - wsa_commons.h, [52](#)
- WSA_RFE0560_ABS_AMP_LOW
 - wsa_commons.h, [52](#)

WSA_RFE0560_ABS_AMP_MEDIUM
 wsa_commons.h, [52](#)

WSA_RFE0560_ABS_AMP_ULOW
 wsa_commons.h, [52](#)

WSA_RFE0560_FREQRES
 wsa_commons.h, [52](#)

WSA_RFE0560_MAX_FREQ
 wsa_commons.h, [52](#)

WSA_RFE0560_MIN_FREQ
 wsa_commons.h, [52](#)

wsa_run_cal_mode
 wsa_api.cpp, [27](#)
 wsa_api.h, [38](#)

wsa_send_command
 wsa_lib.cpp, [61](#)
 wsa_lib.h, [68](#)

wsa_send_query
 wsa_lib.cpp, [62](#)
 wsa_lib.h, [69](#)

wsa_set_antenna
 wsa_api.cpp, [28](#)
 wsa_api.h, [38](#)

wsa_set_bpf
 wsa_api.cpp, [28](#)
 wsa_api.h, [38](#)

wsa_set_freq
 wsa_api.cpp, [28](#)
 wsa_api.h, [39](#)

wsa_set_gain
 wsa_api.cpp, [29](#)
 wsa_api.h, [39](#)

wsa_set_lpf
 wsa_api.cpp, [29](#)
 wsa_api.h, [40](#)

wsa_sock_recv
 wsa_client.cpp, [43](#)
 wsa_client.h, [48](#)

wsa_sock_recv_words
 wsa_client.cpp, [44](#)

wsa_sock_send
 wsa_client.cpp, [44](#)
 wsa_client.h, [49](#)

wsa_socket, [7](#)
 cmd, [7](#)
 data, [7](#)

wsa_start_client
 wsa_client.cpp, [44](#)
 wsa_client.h, [49](#)

wsa_time, [7](#)
 nsec, [8](#)
 sec, [8](#)

wsa_verify_addr
 wsa_client.cpp, [45](#)
 wsa_client.h, [49](#)

wsa_verify_freq
 wsa_api.cpp, [30](#)

WSAGetLastError
 ws-util.cpp, [12](#)
 ws-util.h, [14](#)