

Standard WSA Library

Generated by Doxygen 1.7.4

Thu Dec 1 2011 12:59:43

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	How to use the library . . . . .	1
<b>2</b>	<b>Data Structure Index</b>	<b>1</b>
2.1	Data Structures . . . . .	1
<b>3</b>	<b>File Index</b>	<b>1</b>
3.1	File List . . . . .	1
<b>4</b>	<b>Data Structure Documentation</b>	<b>2</b>
4.1	wsa_descriptor Struct Reference . . . . .	2
4.1.1	Field Documentation . . . . .	2
4.2	wsa_device Struct Reference . . . . .	3
4.2.1	Field Documentation . . . . .	4
4.3	wsa_frame_header Struct Reference . . . . .	4
4.3.1	Field Documentation . . . . .	5
4.4	wsa_resp Struct Reference . . . . .	5
4.4.1	Field Documentation . . . . .	5
4.5	wsa_socket Struct Reference . . . . .	6
4.5.1	Field Documentation . . . . .	6
4.6	wsa_time Struct Reference . . . . .	6
4.6.1	Field Documentation . . . . .	6
<b>5</b>	<b>File Documentation</b>	<b>6</b>
5.1	wsa_error.h File Reference . . . . .	7
5.1.1	Define Documentation . . . . .	9
5.1.2	Function Documentation . . . . .	11
5.2	wsa_lib.cpp File Reference . . . . .	12
5.2.1	Function Documentation . . . . .	12
5.3	wsa_lib.h File Reference . . . . .	19
5.3.1	Define Documentation . . . . .	21
5.3.2	Enumeration Type Documentation . . . . .	21
5.3.3	Function Documentation . . . . .	21
5.4	wsa_lib.txt File Reference . . . . .	27
5.4.1	Detailed Description . . . . .	27

## 1 Introduction

The `wsa_lib` is a library with high level interfaces to a WSA device. It abstracts away the actual low level interface and communication through the connection of choice, and subsequently all the controls or commands to the WSA. It allows you to easily control the WSA4000 through standardized command syntax, such as SCPI, to get WSA status, set gain, set centre frequency, etc., and perform data acquisition.

The `wsa_lib` supports SCPI for control command syntax and VRT for packet.

### 1.1 How to use the library

The `wsa_lib` is designed using mixed C/C++ languages. To use the library, you need to include the header file, [wsa\\_lib.h](#), in files that will use any of its functions to access a WSA, and a link to the `wsa_lib.lib`.

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">wsa_descriptor</a> (This structure stores WSA information )	2
<a href="#">wsa_device</a> (A structure containing the components associate with each WSA device )	3
<a href="#">wsa_frame_header</a> (This structure contains header information related to each frame read by <a href="#">wsa_read_frame()</a> )	4
<a href="#">wsa_resp</a> (This structure contains the response information for each query )	5
<a href="#">wsa_socket</a> (A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition )	6
<a href="#">wsa_time</a> (This structure contains the time information. It is used for the time stamp in a frame header )	6

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">wsa_error.h</a>	7
<a href="#">wsa_lib.cpp</a>	12
<a href="#">wsa_lib.h</a>	19

## 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](#)
- uint32\_t [max\\_sample\\_size](#)
- uint64\_t [max\\_tune\\_freq](#)
- uint64\_t [min\\_tune\\_freq](#)
- uint64\_t [freq\\_resolution](#)
- int [max\\_if\\_gain](#)
- int [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 quantized RF gain setting of the RFE. 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 int [max\\_if\\_gain](#)

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

**4.1.1.7 uint32\_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 int 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.

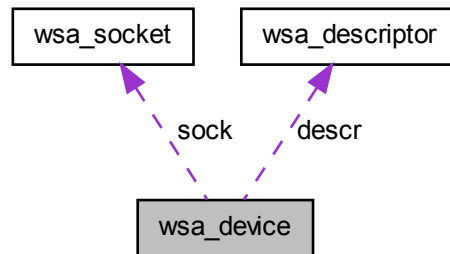
**4.1.1.15 char rfe\_version**

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](#).

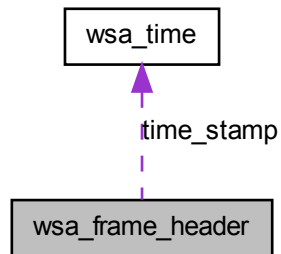
##### 4.2.1.2 struct `wsa_socket` `sock`

The socket structure component of the WSA, used for TCP/IP connection.

#### 4.3 wsa\_frame\_header Struct Reference

This structure contains header information related to each frame read by [wsa\\_read\\_frame\(\)](#).

Collaboration diagram for wsa\_frame\_header:



#### Data Fields

- uint32\_t [sample\\_size](#)
- struct [wsa\\_time](#) [time\\_stamp](#)

#### 4.3.1 Field Documentation

##### 4.3.1.1 uint32\_t sample\_size

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

##### 4.3.1.2 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 [output](#) [MAX\_STR\_LEN]

#### 4.4.1 Field Documentation

##### 4.4.1.1 char output

The char pointer to an output 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 port used for this socket is 37001.

##### 4.5.1.2 SOCKET data

The data socket used for streaming of data. The port used for this socket is 37000.

### 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

- uint32\_t [sec](#)
- uint64\_t [psec](#)

#### 4.6.1 Field Documentation

##### 4.6.1.1 uint64\_t psec

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

##### 4.6.1.2 uint32\_t sec

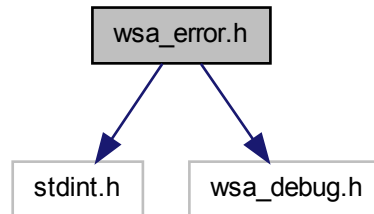
The number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC.

## 5 File Documentation

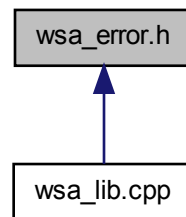


## 5.1 wsa\_error.h File Reference

Include dependency graph for wsa\_error.h:



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



### 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\\_QUERYNORESP](#) (LNEG\_NUM - 9)
- #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_SIZESETFAILED` (LNEG\_NUM - 404)
- `#define WSA_ERR_NOTIQFRAME` (LNEG\_NUM - 405)
- `#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_RESPUNKNOWN` (LNEG\_NUM - 1503)
- `#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_INVRFESSETTING` (LNEG\_NUM - 1607)
- `#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)

### 5.1.1 Define Documentation

- 5.1.1.1 `#define LNEG_NUM` (-10000)
- 5.1.1.2 `#define WSA_ERR_ANNASETFAILED` (LNEG\_NUM - 1602)
- 5.1.1.3 `#define WSA_ERR_CALIBRATESETFAILED` (LNEG\_NUM - 1606)
- 5.1.1.4 `#define WSA_ERR_CMDINVALID` (LNEG\_NUM - 1502)
- 5.1.1.5 `#define WSA_ERR_CMSENDFAILED` (LNEG\_NUM - 1501)
- 5.1.1.6 `#define WSA_ERR_ETHERNETCONNECTFAILED` (LNEG\_NUM - 207)
- 5.1.1.7 `#define WSA_ERR_ETHERNETINITFAILED` (LNEG\_NUM - 209)
- 5.1.1.8 `#define WSA_ERR_ETHERNETNOTAVBL` (LNEG\_NUM - 206)
- 5.1.1.9 `#define WSA_ERR_FILECREATEFAILED` (LNEG\_NUM - 1900)
- 5.1.1.10 `#define WSA_ERR_FILEOPENFAILED` (LNEG\_NUM - 1901)
- 5.1.1.11 `#define WSA_ERR_FILEREADFAILED` (LNEG\_NUM - 1902)
- 5.1.1.12 `#define WSA_ERR_FILEWRITEFAILED` (LNEG\_NUM - 1903)
- 5.1.1.13 `#define WSA_ERR_FILTERSETFAILED` (LNEG\_NUM - 1604)
- 5.1.1.14 `#define WSA_ERR_FREQOUTOFBOUND` (LNEG\_NUM - 601)
- 5.1.1.15 `#define WSA_ERR_FREQSETFAILED` (LNEG\_NUM - 603)
- 5.1.1.16 `#define WSA_ERR_IFGAINSETFAILED` (LNEG\_NUM - 803)
- 5.1.1.17 `#define WSA_ERR_INITFAILED` (LNEG\_NUM - 105)
- 5.1.1.18 `#define WSA_ERR_INVADCCORRVALUE` (LNEG\_NUM - 106)

5.1.1.19 `#define WSA_ERR_INVAMP (LNEG_NUM - 301)`

5.1.1.20 `#define WSA_ERR_INVANTENNAPORT (LNEG_NUM - 1601)`

5.1.1.21 `#define WSA_ERR_INVCALIBRATEMODE (LNEG_NUM - 1605)`

5.1.1.22 `#define WSA_ERR_INVDWELL (LNEG_NUM - 1208)`

5.1.1.23 `#define WSA_ERR_INVFILTERMODE (LNEG_NUM - 1603)`

5.1.1.24 `#define WSA_ERR_INVFREQRES (LNEG_NUM - 602)`

5.1.1.25 `#define WSA_ERR_INVIFGAIN (LNEG_NUM - 802)`

5.1.1.26 `#define WSA_ERR_INVINTFMETHOD (LNEG_NUM - 201)`

5.1.1.27 `#define WSA_ERR_INVIPADDRESS (LNEG_NUM - 2)`

5.1.1.28 `#define WSA_ERR_INVIPHOSTADDRESS (LNEG_NUM - 202)`

5.1.1.29 `#define WSA_ERR_INVNUMBER (LNEG_NUM - 2000)`

5.1.1.30 `#define WSA_ERR_INVNUMFRAMES (LNEG_NUM - 1209)`

5.1.1.31 `#define WSA_ERR_INVREGADDR (LNEG_NUM - 2001)`

5.1.1.32 `#define WSA_ERR_INVRFESETTING (LNEG_NUM - 1607)`

5.1.1.33 `#define WSA_ERR_INVRFGAIN (LNEG_NUM - 801)`

5.1.1.34 `#define WSA_ERR_INVRUNMODE (LNEG_NUM - 1001)`

5.1.1.35 `#define WSA_ERR_INVSAMPLESIZE (LNEG_NUM - 403)`

5.1.1.36 `#define WSA_ERR_INVSTARTRES (LNEG_NUM - 1205)`

5.1.1.37 `#define WSA_ERR_INVSTOPFREQ (LNEG_NUM - 1202)`

5.1.1.38 `#define WSA_ERR_INVSTOPRES (LNEG_NUM - 1206)`

5.1.1.39 `#define WSA_ERR_INVTRIGID (LNEG_NUM - 1201)`

5.1.1.40 `#define WSA_ERR_INVTRIGRANGE (LNEG_NUM - 1207)`

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

5.1.1.42 `#define WSA_ERR_NOCTRLPIPE (LNEG_NUM - 3)`

5.1.1.43 `#define WSA_ERR_NODATABUS (LNEG_NUM - 401)`

5.1.1.44 `#define WSA_ERR_NOTIQFRAME (LNEG_NUM - 405)`

5.1.1.45 `#define WSA_ERR_NOWSA (LNEG_NUM - 1)`

5.1.1.46 `#define WSA_ERR_OPENFAILED (LNEG_NUM - 104)`

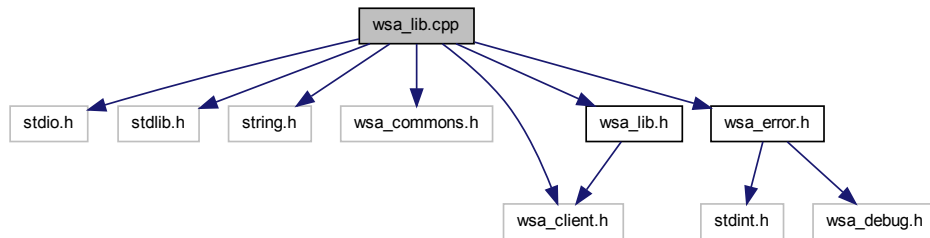
- 5.1.1.47 `#define WSA_ERR_PLLOCKFAILED (LNEG_NUM - 604)`
- 5.1.1.48 `#define WSA_ERR_PRODOBSOLETE (LNEG_NUM - 8)`
- 5.1.1.49 `#define WSA_ERR_QUERYNOESP (LNEG_NUM - 9)`
- 5.1.1.50 `#define WSA_ERR_READFRAMEFAILED (LNEG_NUM - 402)`
- 5.1.1.51 `#define WSA_ERR_RESPUNKOWN (LNEG_NUM - 1503)`
- 5.1.1.52 `#define WSA_ERR_RFGAINSETFAILED (LNEG_NUM - 804)`
- 5.1.1.53 `#define WSA_ERR_SETFAILED (LNEG_NUM - 103)`
- 5.1.1.54 `#define WSA_ERR_SIZESETFAILED (LNEG_NUM - 404)`
- 5.1.1.55 `#define WSA_ERR_SOCKETSETFUPFAILED (LNEG_NUM - 211)`
- 5.1.1.56 `#define WSA_ERR_STARTOOB (LNEG_NUM - 1203)`
- 5.1.1.57 `#define WSA_ERR_STOPOOB (LNEG_NUM - 1204)`
- 5.1.1.58 `#define WSA_ERR_UNKNOWN_ERROR (LNEG_NUM - 2003)`
- 5.1.1.59 `#define WSA_ERR_UNKNOWNFWRVSN (LNEG_NUM - 6)`
- 5.1.1.60 `#define WSA_ERR_UNKNOWNPRODSE (LNEG_NUM - 4)`
- 5.1.1.61 `#define WSA_ERR_UNKNOWNPRODVSN (LNEG_NUM - 5)`
- 5.1.1.62 `#define WSA_ERR_UNKNOWNRFEVSN (LNEG_NUM - 7)`
- 5.1.1.63 `#define WSA_ERR_USBINITFAILED (LNEG_NUM - 205)`
- 5.1.1.64 `#define WSA_ERR_USBNOTAVBL (LNEG_NUM - 203)`
- 5.1.1.65 `#define WSA_ERR_USBOPENFAILED (LNEG_NUM - 204)`
- 5.1.1.66 `#define WSA_ERR_WINSOCKSTARTUPFAILED (LNEG_NUM - 210)`
- 5.1.1.67 `#define WSA_ERR_WSAINUSE (LNEG_NUM - 102)`
- 5.1.1.68 `#define WSA_ERR_WSANOTRDY (LNEG_NUM - 101)`

## 5.1.2 Function Documentation

- 5.1.2.1 `const char* _wsa_get_err_msg ( int16_t err_id )`

## 5.2 wsa\_lib.cpp File Reference

Include dependency graph for wsa\_lib.cpp:



### Functions

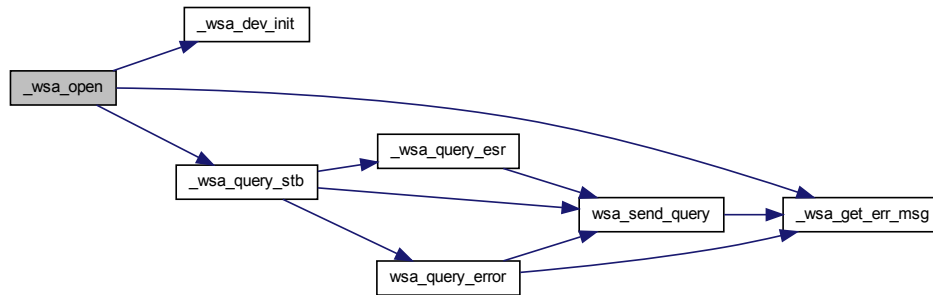
- char \* [wsa\\_query\\_error](#) (struct [wsa\\_device](#) \*dev)
- int16\_t [\\_wsa\\_dev\\_init](#) (struct [wsa\\_device](#) \*dev)
- int16\_t [\\_wsa\\_open](#) (struct [wsa\\_device](#) \*dev)
- int16\_t [\\_wsa\\_query\\_stb](#) (struct [wsa\\_device](#) \*dev, char \*output)
- int16\_t [\\_wsa\\_query\\_esr](#) (struct [wsa\\_device](#) \*dev, char \*output)
- 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\\_read\\_status](#) (struct [wsa\\_device](#) \*dev, char \*output)
- const char \* [wsa\\_get\\_error\\_msg](#) (int16\_t err\_code)
- int16\_t [wsa\\_read\\_frame](#) (struct [wsa\\_device](#) \*dev, struct [wsa\\_frame\\_header](#) \*header, char \*data\_buf, uint32\_t sample\_size, uint32\_t time\_out)
- int32\_t [wsa\\_decode\\_frame](#) (char \*data\_buf, int16\_t \*i\_buf, int16\_t \*q\_buf, uint32\_t sample\_size)

### 5.2.1 Function Documentation

#### 5.2.1.1 int16\_t \_wsa\_dev\_init ( struct wsa\_device \* dev )

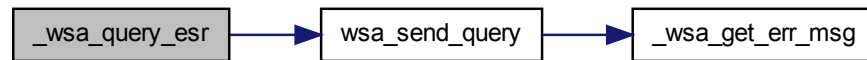
## 5.2.1.2 int16\_t \_wsa\_open ( struct wsa\_device \* dev )

Here is the call graph for this function:



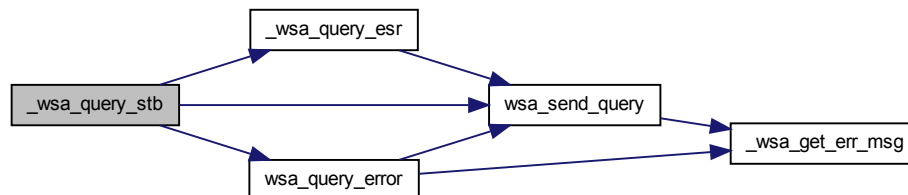
## 5.2.1.3 int16\_t \_wsa\_query\_esr ( struct wsa\_device \* dev, char \* output )

Here is the call graph for this function:



## 5.2.1.4 int16\_t \_wsa\_query\_stb ( struct wsa\_device \* dev, char \* output )

Here is the call graph for this function:



### 5.2.1.5 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.

After successfully connected, this function will also do:

- Check for any errors in WSA
- Gather information for the WSA's descriptor

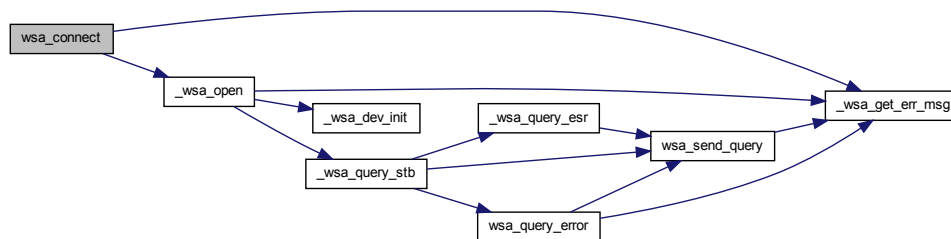
#### Parameters

<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"> <li>• With USB, use: "USB" (check if supported with the WSA version used).</li> <li>• With LAN, use: "TCPIP::<ip address="" of="" port&gt;]".<br="" port,data="" the="" wsa&gt;[::&lt;cmd=""></ip>The ports' number if not entered will be defaulted to: <ul style="list-style-type: none"> <li>– command port: 37001</li> <li>– data port: 37000</li> </ul> </li> </ul> <p>However, if port forwarding method is used to forward different ports to the required ports eventually, then you can enter the ports in the format and the <b>order</b> as specified. Example: "TCPIP::192.168.1.1" or "TCPIP::192.168.1.1::37001,37001"</p>

#### Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



### 5.2.1.6 int32\_t wsa\_decode\_frame ( char \* data\_buf, int16\_t \* i\_buf, int16\_t \* q\_buf, uint32\_t sample\_size )

Decodes the raw **data\_buf** buffer containing frame(s) of I & Q data bytes and returned the I and Q buffers of data with the size determined by the **sample\_size** parameter.

Note: the **data\_buf** size is assumed as **sample\_size** \* 4 bytes per sample



**Parameters**

<i>data_buf</i>	- A char pointer buffer containing the raw I and Q data in bytes to be decoded into separate I and Q buffers. Its size is assumed to be the number of 32-bit <code>sample_size</code> words multiply by 4 (i.e. <code>sizeof(data_buf) = sample_size * 4</code> bytes per sample).
<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <b>sample_size</b> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled, Q data buffer with size specified by the <b>sample_size</b> .
<i>sample_size</i>	- A 32-bit unsigned integer number of {I, Q} sample pairs to be decoded from <b>data_buf</b> . The frame size is limited to a maximum number, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.

**Returns**

The number of samples decoded, or a 16-bit negative number on error.

5.2.1.7 `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.

5.2.1.8 `const char* wsa_get_error_msg ( int16_t err_code )`

Returns a message string associated with the given **err\_code** that is returned from a **wsa\_lib** function.

**Parameters**

<i>err_code</i>	- The negative WSA error code, returned from a WSA function.
-----------------	--

**Returns**

A char pointer to the error message string.

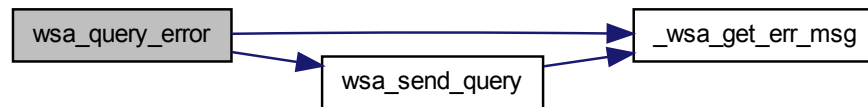
Here is the call graph for this function:



5.2.1.9 `int16_t wsa_list_devs ( char ** wsa_list )`

5.2.1.10 `char * wsa_query_error ( struct wsa_device * dev )`

Here is the call graph for this function:



5.2.1.11 `int16_t wsa_read_frame ( struct wsa_device * dev, struct wsa_frame_header * header, char * data_buf, uint32_t sample_size, uint32_t time_out )`

Reads a frame of data. *Each* frame consists of a header and a buffer of data of length determined by the **sample\_size** parameter (i.e. `sizeof(data_buf) = sample_size * 4` bytes per sample).

Each I and Q samples is 16-bit (2-byte) wide, signed 2-complement. The raw `data_buf` contains alternatively 2-byte Q follows by 2-byte I, so on. In another words, the I & Q samples are distributed in the raw `data_buf` as follow:

```
data_buf = IQIQIQIQ... = <2 bytes I><2 bytes Q><...>
```

The bytes can be decoded, as an example, as follow:

Let takes the first 4 bytes of the `\b data_buf`, for example, then:

```
int16_t I = data_buf[3] << 8 + data_buf[2];
int16_t Q = data_buf[1] << 8 + data_buf[0];
```

And so on for N number of samples:

```
int16_t I[i] = data_buf[i+3] << 8 + data_buf[i+2];
int16_t Q[i] = data_buf[i+1] << 8 + data_buf[i];
```

where `i = 0, 1, 2, ..., (N - 2), (N - 1)`.

Alternatively, the `data_buf` can be passed to [wsa\\_decode\\_frame\(\)](#) to have I and Q splitted up and stored into separate `int16_t` buffers. The [wsa\\_decode\\_frame\(\)](#) function is useful for later needs of decoding the data bytes when a large amount of raw data (multiple frames) has been captured for instance.

## Remarks

This function does not set the **sample\_size** to WSA at each capture in order to minimize the delay between captures. The number of samples per frame must be sent to WSA at least once during the WSA powered on. For example, with SCPI, do:

```
wsa_send_command(dev, "TRACE:IQ:POINTS 1024\n");
```

## Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to <a href="#">wsa_frame_header</a> structure to store information for the frame.
<i>data_buf</i>	- A char pointer buffer to store the raw I and Q data in in bytes. Its size is determined by the number of 32-bit <b>sample_size</b> words multiply by 4 (i.e. <code>sizeof(data_buf) = sample_size * 4</code> bytes per sample, which is automatically done by the function).
<i>sample_size</i>	- A 32-bit unsigned integer sample size (i.e. number of {I, Q} sample pairs) per data frame to be captured. The size is limited to a maximum number, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.
<i>time_out</i>	- The time, in milliseconds, to wait for a packet from a WSA before time out.

**Returns**

A 4-bit frame count number that starts at 0, or a 16-bit negative number on error.

5.2.1.12 `int16_t wsa_read_status ( struct wsa_device * dev, char * output )`

Query the status of the WSA box for any event and store the output response(s) in the **output** parameter.

**Remarks**

This function is equivalent to the SCPI command "`*STB?`".

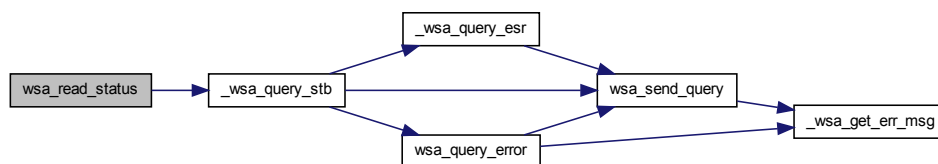
**Parameters**

<i>dev</i>	
<i>output</i>	- a char pointer to the output result message of the query

**Returns**

0 if successfully queried, or a negative number upon errors.

Here is the call graph for this function:



5.2.1.13 `int16_t wsa_send_command ( struct wsa_device * dev, char * command )`

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\(\)](#).

**Remarks**

To send query command, use [wsa\\_send\\_query\(\)](#) instead.

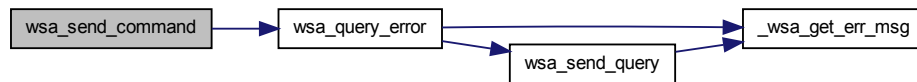
**Parameters**

<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 <a href="#">wsa_connect()</a>

**Returns**

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

Here is the call graph for this function:



#### 5.2.1.14 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.

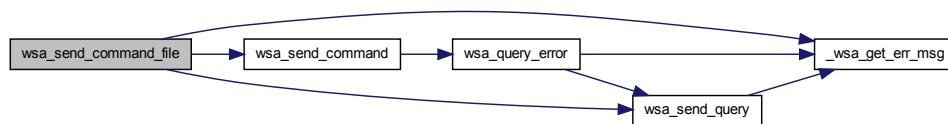
**Parameters**

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



5.2.1.15 `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\(\)](#) (ex. SCPI).

#### 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 <a href="#">wsa_connect()</a> .

#### Returns

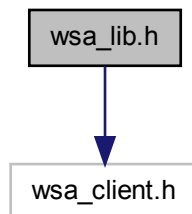
The result stored in a [wsa\\_resp](#) struct format.

Here is the call graph for this function:

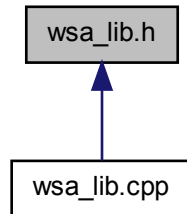


## 5.3 wsa\_lib.h File Reference

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\\_read\\_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\\_MED](#), [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_read_status` (struct `wsa_device` \*dev, char \*output)
- `const char * wsa_get_error_msg` (int16\_t err\_code)
- `int16_t wsa_read_frame` (struct `wsa_device` \*dev, struct `wsa_frame_header` \*header, char \*data\_buf, uint32\_t sample\_size, uint32\_t time\_out)
- `int32_t wsa_decode_frame` (char \*data\_buf, int16\_t \*i\_buf, int16\_t \*q\_buf, uint32\_t sample\_size)

### 5.3.1 Define Documentation

5.3.1.1 `#define FALSE 0`

5.3.1.2 `#define NUM_RF_GAINS 5`

5.3.1.3 `#define SCPI "SCPI"`

5.3.1.4 `#define TRUE 1`

### 5.3.2 Enumeration Type Documentation

5.3.2.1 `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\_MED** Medium RF amplification.

**WSA\_GAIN\_LOW** Low RF amplification.

**WSA\_GAIN\_VLOW** Very low RF amplification.

### 5.3.3 Function Documentation

5.3.3.1 `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.

After successfully connected, this function will also do:

- Check for any errors in WSA
- Gather information for the WSA's descriptor

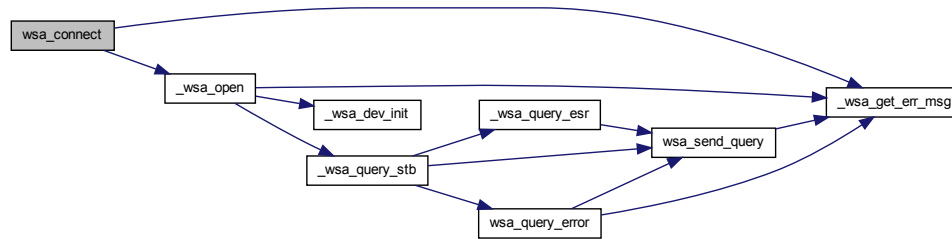
## Parameters

<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>	<p>- A char pointer to store the interface method to the WSA.</p> <p>Possible methods:</p> <ul style="list-style-type: none"> <li>• With USB, use: "USB" (check if supported with the WSA version used).</li> <li>• With LAN, use: "TCPIP::<ip <ul="" address="" be="" defaulted="" entered="" if="" not="" number="" of="" port&gt;]".="" port,data="" ports'="" the="" to:="" will="" wsa&gt;[:&lt;cmd=""> <li>– command port: 37001</li> <li>– data port: 37000</li> </ip></li></ul> <p>However, if port forwarding method is used to forward different ports to the required ports eventually, then you can enter the ports in the format and the <b>order</b> as specified. Example: "TCPIP::192.168.1.1" or "TCPIP::192.168.1.1::37001,37001"</p>

### Returns

0 on success, or a negative number on error.

Here is the call graph for this function:



#### 5.3.3.2 int32\_t wsa\_decode\_frame ( char \* data\_buf, int16\_t \* i\_buf, int16\_t \* q\_buf, uint32\_t sample\_size )

Decodes the raw **data\_buf** buffer containing frame(s) of I & Q data bytes and returned the I and Q buffers of data with the size determined by the **sample\_size** parameter.

Note: the **data\_buf** size is assumed as **sample\_size** \* 4 bytes per sample

### Parameters

<i>data_buf</i>	- A char pointer buffer containing the raw I and Q data in bytes to be decoded into separate I and Q buffers. Its size is assumed to be the number of 32-bit sample_size words multiply by 4 (i.e. sizeof(data_buf) = sample_size * 4 bytes per sample).
<i>i_buf</i>	- A 16-bit signed integer pointer for the unscaled, I data buffer with size specified by the <b>sample_size</b> .
<i>q_buf</i>	- A 16-bit signed integer pointer for the unscaled, Q data buffer with size specified by the <b>sample_size</b> .



<i>sample_size</i>	- A 32-bit unsigned integer number of {I, Q} sample pairs to be decoded from <b>data_buf</b> . The frame size is limited to a maximum number, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.
--------------------	--

**Returns**

The number of samples decoded, or a 16-bit negative number on error.

5.3.3.3 `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.

5.3.3.4 `const char* wsa_get_error_msg ( int16_t err_code )`

Returns a message string associated with the given **err\_code** that is returned from a **wsa\_lib** function.

**Parameters**

<i>err_code</i>	- The negative WSA error code, returned from a WSA function.
-----------------	--

**Returns**

A char pointer to the error message string.

Here is the call graph for this function:

5.3.3.5 `int16_t wsa_list_devs ( char ** wsa_list )`5.3.3.6 `int16_t wsa_read_frame ( struct wsa_device * dev, struct wsa_frame_header * header, char * data_buf, uint32_t sample_size, uint32_t time_out )`

Reads a frame of data. *Each* frame consists of a header and a buffer of data of length determined by the **sample\_size** parameter (i.e. `sizeof(data_buf) = sample_size * 4` bytes per sample).

Each I and Q samples is 16-bit (2-byte) wide, signed 2-complement. The raw `data_buf` contains alternatively 2-byte Q follows by 2-byte I, so on. In another words, the I & Q samples are distributed in the raw `data_buf` as follow:

```
data_buf = IQIQIQIQ... = <2 bytes I><2 bytes Q><...>
```

The bytes can be decoded, as an example, as follow:

Let takes the first 4 bytes of the `\b data_buf`, for example, then:

```
int16_t I = data_buf[3] << 8 + data_buf[2];
int16_t Q = data_buf[1] << 8 + data_buf[0];
```

And so on for N number of samples:

```
int16_t I[i] = data_buf[i+3] << 8 + data_buf[i+2];
int16_t Q[i] = data_buf[i+1] << 8 + data_buf[i];
```

where  $i = 0, 1, 2, \dots, (N - 2), (N - 1)$ .

Alternatively, the `data_buf` can be passed to [wsa\\_decode\\_frame\(\)](#) to have I and Q splitted up and stored into separate `int16_t` buffers. The [wsa\\_decode\\_frame\(\)](#) function is useful for later needs of decoding the data bytes when a large amount of raw data (multiple frames) has been captured for instance.

### Remarks

This function does not set the **sample\_size** to WSA at each capture in order to minimize the delay between captures. The number of samples per frame must be sent to WSA at least once during the WSA powered on. For example, with SCPI, do:

```
wsa_send_command(dev, "TRACE:IQ:POINTS 1024\n");
```

### Parameters

<i>dev</i>	- A pointer to the WSA device structure.
<i>header</i>	- A pointer to <a href="#">wsa_frame_header</a> structure to store information for the frame.
<i>data_buf</i>	- A char pointer buffer to store the raw I and Q data in in bytes. Its size is determined by the number of 32-bit <b>sample_size</b> words multiply by 4 (i.e. <code>sizeof(data_buf) = sample_size * 4</code> bytes per sample, which is automatically done by the function).
<i>sample_size</i>	- A 32-bit unsigned integer sample size (i.e. number of {I, Q} sample pairs) per data frame to be captured. The size is limited to a maximum number, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.
<i>time_out</i>	- The time, in milliseconds, to wait for a packet from a WSA before time out.

### Returns

A 4-bit frame count number that starts at 0, or a 16-bit negative number on error.

#### 5.3.3.7 int16\_t wsa\_read\_status ( struct wsa\_device \* dev, char \* output )

Query the status of the WSA box for any event and store the output response(s) in the **output** parameter.

### Remarks

This function is equivalent to the SCPI command "`*STB?`".

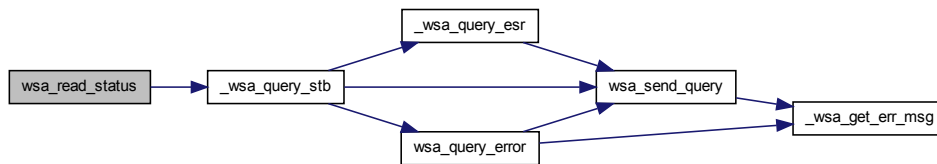
**Parameters**

<i>dev</i>	
<i>output</i>	- a char pointer to the output result message of the query

**Returns**

0 if successfully queried, or a negative number upon errors.

Here is the call graph for this function:



### 5.3.3.8 int16\_t wsa\_send\_command ( struct wsa\_device \* dev, char \* command )

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\(\)](#).

**Remarks**

To send query command, use [wsa\\_send\\_query\(\)](#) instead.

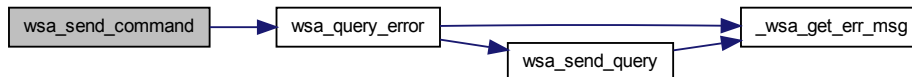
**Parameters**

<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 <a href="#">wsa_connect()</a>

**Returns**

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

Here is the call graph for this function:



### 5.3.3.9 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.

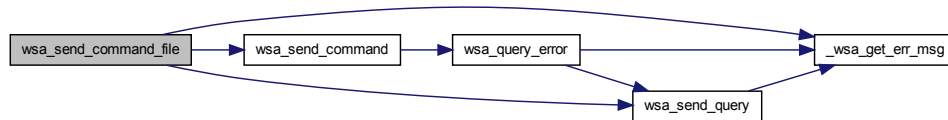
#### Parameters

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



### 5.3.3.10 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\(\)](#) (ex. SCPI).

#### 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 <a href="#">wsa_connect()</a> .

#### Returns

The result stored in a [wsa\\_resp](#) struct format.

Here is the call graph for this function:



## 5.4 wsa\_lib.txt File Reference

Contain some code documents for [wsa\\_lib.h](#).

### 5.4.1 Detailed Description

## Index

`_wsa_dev_init`  
    [wsa\\_lib.cpp, 12](#)  
`_wsa_get_err_msg`  
    [wsa\\_error.h, 11](#)  
`_wsa_open`  
    [wsa\\_lib.cpp, 12](#)  
`_wsa_query_esr`  
    [wsa\\_lib.cpp, 13](#)  
`_wsa_query_stb`  
    [wsa\\_lib.cpp, 13](#)  
  
`abs_max_amp`  
    [wsa\\_descriptor, 2](#)  
  
`cmd`  
    [wsa\\_socket, 6](#)  
  
`data`  
    [wsa\\_socket, 6](#)  
`descr`  
    [wsa\\_device, 4](#)  
  
`FALSE`  
    [wsa\\_lib.h, 21](#)  
`freq_resolution`  
    [wsa\\_descriptor, 2](#)  
`fw_version`  
    [wsa\\_descriptor, 2](#)  
  
`inst_bw`  
    [wsa\\_descriptor, 2](#)  
`intf_type`  
    [wsa\\_descriptor, 2](#)  
  
`LNEG_NUM`  
    [wsa\\_error.h, 9](#)  
  
`max_if_gain`  
    [wsa\\_descriptor, 2](#)  
`max_sample_size`  
    [wsa\\_descriptor, 3](#)  
`max_tune_freq`  
    [wsa\\_descriptor, 3](#)  
`min_if_gain`  
    [wsa\\_descriptor, 3](#)  
`min_tune_freq`  
    [wsa\\_descriptor, 3](#)  
  
`NUM_RF_GAINS`  
    [wsa\\_lib.h, 21](#)

`output`  
    [wsa\\_resp, 5](#)  
  
`prod_name`  
    [wsa\\_descriptor, 3](#)  
`prod_serial`  
    [wsa\\_descriptor, 3](#)  
`prod_version`  
    [wsa\\_descriptor, 3](#)  
`psec`  
    [wsa\\_time, 6](#)  
  
`rfe_name`  
    [wsa\\_descriptor, 3](#)  
`rfe_version`  
    [wsa\\_descriptor, 3](#)  
  
`sample_size`  
    [wsa\\_frame\\_header, 5](#)  
`SCPI`  
    [wsa\\_lib.h, 21](#)  
`sec`  
    [wsa\\_time, 6](#)  
`sock`  
    [wsa\\_device, 4](#)  
`status`  
    [wsa\\_resp, 5](#)  
  
`time_stamp`  
    [wsa\\_frame\\_header, 5](#)  
`TRUE`  
    [wsa\\_lib.h, 21](#)  
  
`WSA_GAIN_HIGH`  
    [wsa\\_lib.h, 21](#)  
`WSA_GAIN_LOW`  
    [wsa\\_lib.h, 21](#)  
`WSA_GAIN_MED`  
    [wsa\\_lib.h, 21](#)  
`WSA_GAIN_VLOW`  
    [wsa\\_lib.h, 21](#)  
`wsa_lib.h`  
    [WSA\\_GAIN\\_HIGH, 21](#)  
    [WSA\\_GAIN\\_LOW, 21](#)  
    [WSA\\_GAIN\\_MED, 21](#)  
    [WSA\\_GAIN\\_VLOW, 21](#)  
`wsa_connect`  
    [wsa\\_lib.cpp, 13](#)  
    [wsa\\_lib.h, 21](#)  
`wsa_decode_frame`

`wsa_lib.cpp`, 14  
`wsa_lib.h`, 22  
`wsa_descriptor`, 2  
  `abs_max_amp`, 2  
  `freq_resolution`, 2  
  `fw_version`, 2  
  `inst_bw`, 2  
  `intf_type`, 2  
  `max_if_gain`, 2  
  `max_sample_size`, 3  
  `max_tune_freq`, 3  
  `min_if_gain`, 3  
  `min_tune_freq`, 3  
  `prod_name`, 3  
  `prod_serial`, 3  
  `prod_version`, 3  
  `rfe_name`, 3  
  `rfe_version`, 3  
`wsa_device`, 3  
  `descr`, 4  
  `sock`, 4  
`wsa_disconnect`  
  `wsa_lib.cpp`, 15  
  `wsa_lib.h`, 23  
`WSA_ERR_ANTENNASETFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_CALIBRATESETFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_CMDINVALID`  
  `wsa_error.h`, 9  
`WSA_ERR_CMDSENDFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_ETHERNETCONNECTFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_ETHERNETINITFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_ETHERNETNOTAVBL`  
  `wsa_error.h`, 9  
`WSA_ERR_FILECREATEFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_FILEOPENFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_FILEREADFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_FILEWRITEFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_FILTERSETFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_FREQOUTOFBOUND`  
  `wsa_error.h`, 9  
`WSA_ERR_FREQSETFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_IFGAINSETFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_INITFAILED`  
  `wsa_error.h`, 9  
`WSA_ERR_INVADCCORRVALUE`  
  `wsa_error.h`, 9  
`WSA_ERR_INVAMP`  
  `wsa_error.h`, 9  
`WSA_ERR_INVANTENNAPORT`  
  `wsa_error.h`, 10  
`WSA_ERR_INVCALIBRATEMODE`  
  `wsa_error.h`, 10  
`WSA_ERR_INVDWELL`  
  `wsa_error.h`, 10  
`WSA_ERR_INVFILTERMODE`  
  `wsa_error.h`, 10  
`WSA_ERR_INVFREQRES`  
  `wsa_error.h`, 10  
`WSA_ERR_INVIFGAIN`  
  `wsa_error.h`, 10  
`WSA_ERR_INVINTFMETHOD`  
  `wsa_error.h`, 10  
`WSA_ERR_INVIPADDRESS`  
  `wsa_error.h`, 10  
`WSA_ERR_INVIPHOSTADDRESS`  
  `wsa_error.h`, 10  
`WSA_ERR_INVNUMBER`  
  `wsa_error.h`, 10  
`WSA_ERR_INVNUMFRAMES`  
  `wsa_error.h`, 10  
`WSA_ERR_INVREGADDR`  
  `wsa_error.h`, 10  
`WSA_ERR_INVRFESETTING`  
  `wsa_error.h`, 10  
`WSA_ERR_INVRFGAIN`  
  `wsa_error.h`, 10  
`WSA_ERR_INVRUNMODE`  
  `wsa_error.h`, 10  
`WSA_ERR_INVSAMPLESIZE`  
  `wsa_error.h`, 10  
`WSA_ERR_INVSTARTRES`  
  `wsa_error.h`, 10  
`WSA_ERR_INVSTOPFREQ`  
  `wsa_error.h`, 10  
`WSA_ERR_INVSTOPRES`  
  `wsa_error.h`, 10  
`WSA_ERR_INVTRIGID`  
  `wsa_error.h`, 10  
`WSA_ERR_INVTRIGRANGE`  
  `wsa_error.h`, 10  
`WSA_ERR_MALLOCFAILED`  
  `wsa_error.h`, 10

WSA\_ERR\_NOCTRLPIPE  
    wsa\_error.h, 10

WSA\_ERR\_NODATABUS  
    wsa\_error.h, 10

WSA\_ERR\_NOTIQFRAME  
    wsa\_error.h, 10

WSA\_ERR\_NOWSA  
    wsa\_error.h, 10

WSA\_ERR\_OPENFAILED  
    wsa\_error.h, 10

WSA\_ERR\_PLLOCKFAILED  
    wsa\_error.h, 10

WSA\_ERR\_PRODOBSOLETE  
    wsa\_error.h, 11

WSA\_ERR\_QUERYNORESP  
    wsa\_error.h, 11

WSA\_ERR\_READFRAMEFAILED  
    wsa\_error.h, 11

WSA\_ERR\_RESPUNKOWN  
    wsa\_error.h, 11

WSA\_ERR\_RFGAINSETFAILED  
    wsa\_error.h, 11

WSA\_ERR\_SETFAILED  
    wsa\_error.h, 11

WSA\_ERR\_SIZESETFAILED  
    wsa\_error.h, 11

WSA\_ERR\_SOCKETSETFUPFAILED  
    wsa\_error.h, 11

WSA\_ERR\_STARTOOB  
    wsa\_error.h, 11

WSA\_ERR\_STOPOOB  
    wsa\_error.h, 11

WSA\_ERR\_UNKNOWN\_ERROR  
    wsa\_error.h, 11

WSA\_ERR\_UNKNOWNFWRVSN  
    wsa\_error.h, 11

WSA\_ERR\_UNKNOWNPRODSE  
    wsa\_error.h, 11

WSA\_ERR\_UNKNOWNPRODSN  
    wsa\_error.h, 11

WSA\_ERR\_UNKNOWNRFEVSN  
    wsa\_error.h, 11

WSA\_ERR\_USBINITFAILED  
    wsa\_error.h, 11

WSA\_ERR\_USBNOTAVBL  
    wsa\_error.h, 11

WSA\_ERR\_USBOPENFAILED  
    wsa\_error.h, 11

WSA\_ERR\_WINSOCKSTARTUPFAILED  
    wsa\_error.h, 11

WSA\_ERR\_WSAINUSE  
    wsa\_error.h, 11

WSA\_ERR\_WSANOTRDY  
    wsa\_error.h, 11

wsa\_error.h, 7

    \_wsa\_get\_err\_msg, 11

    LNEG\_NUM, 9

    WSA\_ERR\_ANTENNASETFAILED, 9

    WSA\_ERR\_CALIBRATESETFAILED, 9

    WSA\_ERR\_CMDINVALID, 9

    WSA\_ERR\_CMDSENDFAILED, 9

    WSA\_ERR\_ETHERNETCONNECTFAILED, 9

    WSA\_ERR\_ETHERNETINITFAILED, 9

    WSA\_ERR\_ETHERNETNOTAVBL, 9

    WSA\_ERR\_FILECREATEFAILED, 9

    WSA\_ERR\_FILEOPENFAILED, 9

    WSA\_ERR\_FILEREADFAILED, 9

    WSA\_ERR\_FILEWRITEFAILED, 9

    WSA\_ERR\_FILTERSETFAILED, 9

    WSA\_ERR\_FREQOUTOFBOUND, 9

    WSA\_ERR\_FREQSETFAILED, 9

    WSA\_ERR\_IFGAINSETFAILED, 9

    WSA\_ERR\_INITFAILED, 9

    WSA\_ERR\_INVADCCORRVALUE, 9

    WSA\_ERR\_INVAMP, 9

    WSA\_ERR\_INVANTENNAPORT, 10

    WSA\_ERR\_INVCALIBRATEMODE, 10

    WSA\_ERR\_INVDWELL, 10

    WSA\_ERR\_INVFILTERMODE, 10

    WSA\_ERR\_INVFREQRES, 10

    WSA\_ERR\_INVIFGAIN, 10

    WSA\_ERR\_INVINTFMETHOD, 10

    WSA\_ERR\_INVIPADDRESS, 10

    WSA\_ERR\_INVIPHOSTADDRESS, 10

    WSA\_ERR\_INVNUMBER, 10

    WSA\_ERR\_INVNUMFRAMES, 10

    WSA\_ERR\_INVREGADDR, 10

    WSA\_ERR\_INVRFESETTING, 10

    WSA\_ERR\_INVRFGAIN, 10

    WSA\_ERR\_INVRUNMODE, 10

    WSA\_ERR\_INVSAMPLESIZE, 10

    WSA\_ERR\_INVSTARTRES, 10

    WSA\_ERR\_INVSTOPFREQ, 10

    WSA\_ERR\_INVSTOPRES, 10

    WSA\_ERR\_INVTRIGID, 10

    WSA\_ERR\_INVTRIGRANGE, 10

    WSA\_ERR\_MALLOCFAILED, 10

    WSA\_ERR\_NOCTRLPIPE, 10

    WSA\_ERR\_NODATABUS, 10

    WSA\_ERR\_NOTIQFRAME, 10

    WSA\_ERR\_NOWSA, 10

    WSA\_ERR\_OPENFAILED, 10

    WSA\_ERR\_PLLOCKFAILED, 10

    WSA\_ERR\_PRODOBSOLETE, 11



- WSA\_ERR\_QUERYNORESP, [11](#)
- WSA\_ERR\_READFRAMEFAILED, [11](#)
- WSA\_ERR\_RESPUNKNOWN, [11](#)
- WSA\_ERR\_RFGAINSETFAILED, [11](#)
- WSA\_ERR\_SETFAILED, [11](#)
- WSA\_ERR\_SIZESETFAILED, [11](#)
- WSA\_ERR\_SOCKETSETFUPFAILED, [11](#)
- WSA\_ERR\_STARTOOB, [11](#)
- WSA\_ERR\_STOPOOB, [11](#)
- WSA\_ERR\_UNKNOWN\_ERROR, [11](#)
- WSA\_ERR\_UNKNOWNFWRVSN, [11](#)
- WSA\_ERR\_UNKNOWNPRODSE, [11](#)
- WSA\_ERR\_UNKNOWNPRODVS, [11](#)
- WSA\_ERR\_UNKNOWNRFEVSN, [11](#)
- WSA\_ERR\_USBINITFAILED, [11](#)
- WSA\_ERR\_USBNOTAVBL, [11](#)
- WSA\_ERR\_USBOPENFAILED, [11](#)
- WSA\_ERR\_WINSOCKSTARTUPFAILED, [11](#)
- WSA\_ERR\_WSAINUSE, [11](#)
- WSA\_ERR\_WSANOTRDY, [11](#)
- wsa\_frame\_header, [4](#)
  - sample\_size, [5](#)
  - time\_stamp, [5](#)
- wsa\_gain
  - wsa\_lib.h, [21](#)
- wsa\_get\_error\_msg
  - wsa\_lib.cpp, [15](#)
  - wsa\_lib.h, [23](#)
- wsa\_lib.cpp, [12](#)
  - \_wsa\_dev\_init, [12](#)
  - \_wsa\_open, [12](#)
  - \_wsa\_query\_esr, [13](#)
  - \_wsa\_query\_stb, [13](#)
  - wsa\_connect, [13](#)
  - wsa\_decode\_frame, [14](#)
  - wsa\_disconnect, [15](#)
  - wsa\_get\_error\_msg, [15](#)
  - wsa\_list\_devs, [15](#)
  - wsa\_query\_error, [16](#)
  - wsa\_read\_frame, [16](#)
  - wsa\_read\_status, [17](#)
  - wsa\_send\_command, [17](#)
  - wsa\_send\_command\_file, [18](#)
  - wsa\_send\_query, [18](#)
- wsa\_lib.h, [19](#)
  - FALSE, [21](#)
  - NUM\_RF\_GAINS, [21](#)
  - SCPI, [21](#)
  - TRUE, [21](#)
  - wsa\_connect, [21](#)
  - wsa\_decode\_frame, [22](#)
  - wsa\_disconnect, [23](#)
  - wsa\_gain, [21](#)
  - wsa\_get\_error\_msg, [23](#)
  - wsa\_list\_devs, [23](#)
  - wsa\_read\_frame, [23](#)
  - wsa\_read\_status, [24](#)
  - wsa\_send\_command, [25](#)
  - wsa\_send\_command\_file, [25](#)
  - wsa\_send\_query, [26](#)
- wsa\_lib.txt, [27](#)
- wsa\_list\_devs
  - wsa\_lib.cpp, [15](#)
  - wsa\_lib.h, [23](#)
- wsa\_query\_error
  - wsa\_lib.cpp, [16](#)
- wsa\_read\_frame
  - wsa\_lib.cpp, [16](#)
  - wsa\_lib.h, [23](#)
- wsa\_read\_status
  - wsa\_lib.cpp, [17](#)
  - wsa\_lib.h, [24](#)
- wsa\_resp, [5](#)
  - output, [5](#)
  - status, [5](#)
- wsa\_send\_command
  - wsa\_lib.cpp, [17](#)
  - wsa\_lib.h, [25](#)
- wsa\_send\_command\_file
  - wsa\_lib.cpp, [18](#)
  - wsa\_lib.h, [25](#)
- wsa\_send\_query
  - wsa\_lib.cpp, [18](#)
  - wsa\_lib.h, [26](#)
- wsa\_socket, [6](#)
  - cmd, [6](#)
  - data, [6](#)
- wsa\_time, [6](#)
  - psec, [6](#)
  - sec, [6](#)