

## Standard WSA API Library

Generated by Doxygen 1.7.4

Mon Aug 29 2011 16:15:30

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Limitations in v1.0:	1
1.2	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>2</b>
3.1	File List	2
<b>4</b>	<b>Data Structure Documentation</b>	<b>2</b>
4.1	wsa_descriptor Struct Reference	2
4.1.1	Field Documentation	3
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	6
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>7</b>
5.1	ReadMe.txt File Reference	7
5.1.1	Variable Documentation	7
5.2	wsa_api.cpp File Reference	7
5.2.1	Function Documentation	8
5.3	wsa_api.h File Reference	15
5.3.1	Enumeration Type Documentation	17
5.3.2	Function Documentation	17
5.4	wsa_error.h File Reference	24
5.4.1	Define Documentation	26
5.4.2	Function Documentation	27

5.5	<a href="#">wsa_lib.txt File Reference</a>	27
5.5.1	<a href="#">Detailed Description</a>	28

## 1 Introduction

This documentation, compiled using Doxygen, describes in details the `wsa_api` library. The `wsa_api` provides functions to set/get particular settings or acquire 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, it decodes any responses or packet coming back from the WSA through the `wsa_lib`. Thus, the API helps to abstract away SCPI syntax from the user.

Data frames passing back from the `wsa_lib` are in VRT format. This API will extract the information and the actual data frames within the VRT packet and makes them available in structures and buffers for users.

### 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.

### 1.2 How to use the library

The `wsa_api` is designed using mixed C/C++ languages. To use the library, you need to include the header file, [wsa\\_api.h](#), in files that will use any of its functions to access a WSA, and a link to the `wsa_api.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 <code>wsa_get_frame()</code> )	4
<a href="#">wsa_resp</a> (This structure contains the response information for each query )	6
<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_api.cpp</a>	7
<a href="#">wsa_api.h</a>	15
<a href="#">wsa_error.h</a>	24

## 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.

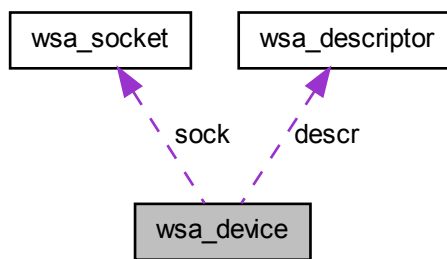
##### 4.1.1.11 char rfe\_version

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](#).

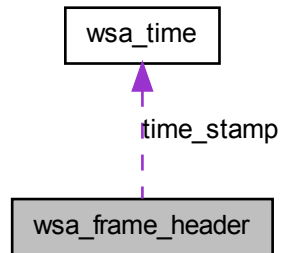
#### 4.2.1.2 struct [wsa\\_socket](#) `sock`

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

## 4.3 wsa\_frame\_header Struct Reference

This structure contains header information related to each frame read by `wsa_get_frame()`.

Collaboration diagram for wsa\_frame\_header:



#### Data Fields

- char [prod\\_serial](#) [20]
- uint64\_t [freq](#)
- char [gain](#) [10]
- uint32\_t [sample\\_size](#)
- struct [wsa\\_time](#) [time\\_stamp](#)

#### 4.3.1 Field Documentation

##### 4.3.1.1 uint64\_t [freq](#)

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

##### 4.3.1.2 char [gain](#)

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

##### 4.3.1.3 char [prod\\_serial](#)

WSA product version number.

##### 4.3.1.4 uint32\_t [sample\\_size](#)

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

##### 4.3.1.5 struct [wsa\\_time](#) [time\\_stamp](#)

The time when a data frame capture begins, stored in [wsa\\_time](#) structure.

#### 4.4 wsa\_resp Struct Reference

This structure contains the response information for each query.

#### 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 ReadMe.txt File Reference

#### Variables

- and information about the [platforms](#)
- and information about the [configurations](#)
- and information about the and project features selected with the Application Wizard
- wsa4000\_cli.cpp This is the main application source file
- Other standard [files](#)

#### 5.1.1 Variable Documentation

##### 5.1.1.1 and information about the [configurations](#)

##### 5.1.1.2 and information about the and project features selected with the Application Wizard

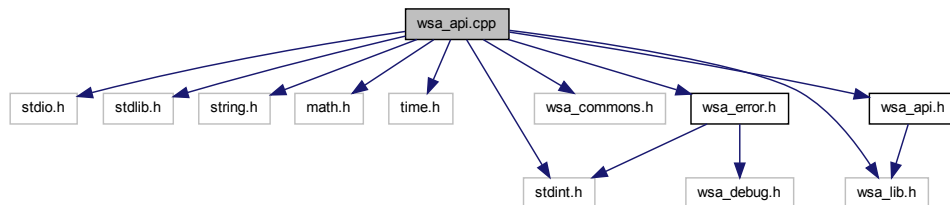
wsa4000\_cli.cpp This is the main application source file

Other standard files

##### 5.1.1.3 and information about the [platforms](#)

### 5.2 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)

### 5.2.1 Function Documentation

#### 5.2.1.1 `int16_t wsa_check_addr ( char * ip_addr )`

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

##### Parameters

<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:



#### 5.2.1.2 `int16_t wsa_check_cal_mode ( struct wsa_device * dev )`

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

##### Parameters

<code>dev</code> - 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.

**5.2.1.3 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

**5.2.1.4 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 <b>wsa_gain</b> 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.

**5.2.1.5 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.

**5.2.1.6 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.

**5.2.1.7 int64\_t wsa\_get\_freq ( struct wsa\_device \* dev )**

Retrieves the center frequency that the WSA is running at.

**Parameters**

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

**Returns**

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

**5.2.1.8 wsa\_gain wsa\_get\_gain ( struct wsa\_device \* dev )**

Gets the current gain setting of the WSA.

**Parameters**

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

**Returns**

The gain setting of wsa\_gain type, or a negative number on error.

**5.2.1.9 int16\_t wsa\_get\_lpf ( struct wsa\_device \* dev )**

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

**Parameters**

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

**Returns**

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

**5.2.1.10 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.

5.2.1.11 `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.

5.2.1.12 `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"> <li>• With LAN, use: "TCPIP::<ip address="" of="" the="" wsa="">::HISLIP"</ip></li> <li>• With USB, use: "USB" (check if supported with the WSA version used).</li> </ul>

**Returns**

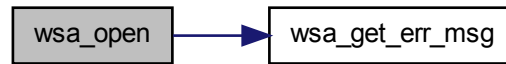
0 on success, or a negative number on error.

**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:



**5.2.1.13** `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 <a href="#">wsa_frame_header</a> 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, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.

#### Returns

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

**5.2.1.14** `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.

5.2.1.15 `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. <b>Note:</b> When calibration mode is enabled through <a href="#">wsa_run_cal_mode()</a> , 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.

5.2.1.16 `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.

5.2.1.17 `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

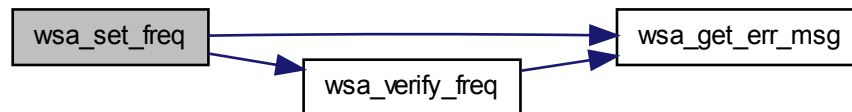
**Returns**

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:



#### 5.2.1.18 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 wsa_gain to set for WSA. Valid gain settings are HIGH, MEDIUM, LOW, ULOW.

**Returns**

0 on success, or a negative number on error.

#### 5.2.1.19 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.



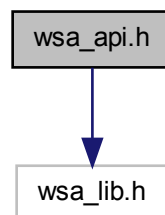
5.2.1.20 `int16_t wsa_verify_freq ( struct wsa_device * dev, uint64_t freq )`

Here is the call graph for this function:

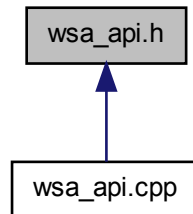


### 5.3 wsa\_api.h File Reference

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](#) }

#### 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)

### 5.3.1 Enumeration Type Documentation

#### 5.3.1.1 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.

### 5.3.2 Function Documentation

#### 5.3.2.1 `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:



#### 5.3.2.2 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.

#### 5.3.2.3 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

#### 5.3.2.4 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 <b>wsa_gain</b> 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.

#### 5.3.2.5 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.

#### 5.3.2.6 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.

#### 5.3.2.7 int64\_t wsa\_get\_freq ( struct wsa\_device \* dev )

Retrieves the center frequency that the WSA is running at.

##### Parameters

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

##### Returns

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

#### 5.3.2.8 wsa\_gain wsa\_get\_gain ( struct wsa\_device \* dev )

Gets the current gain setting of the WSA.

##### Parameters

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

##### Returns

The gain setting of wsa\_gain type, or a negative number on error.

#### 5.3.2.9 int16\_t wsa\_get\_lpf ( struct wsa\_device \* dev )

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

**Parameters**

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

**Returns**

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

**5.3.2.10 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.

**5.3.2.11 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**

<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.

**5.3.2.12 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"> <li>• With LAN, use: "TCP/IP::&lt;Ip address of the WSA&gt;::HISLIP"</li> <li>• With USB, use: "USB" (check if supported with the WSA version used).</li> </ul>

**Returns**

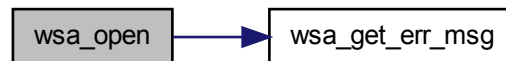
0 on success, or a negative number on error.

**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:



**5.3.2.13** `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 <a href="#">wsa_frame_header</a> 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, <b>max_sample_size</b> , listed in the <a href="#">wsa_descriptor</a> structure.

**Returns**

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

## 5.3.2.14 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.

## 5.3.2.15 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. <b>Note:</b> When calibration mode is enabled through <a href="#">wsa_run_cal_mode()</a> , 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.

## 5.3.2.16 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.

## 5.3.2.17 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

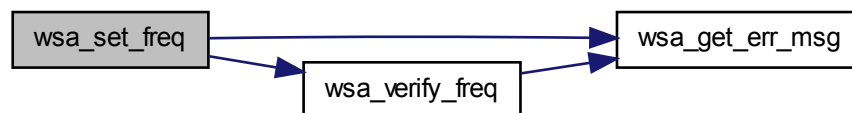
**Returns**

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:



#### 5.3.2.18 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 <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.

#### 5.3.2.19 int16\_t wsa\_set\_lpf ( struct wsa\_device \* dev, uint8\_t option )

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

**Parameters**

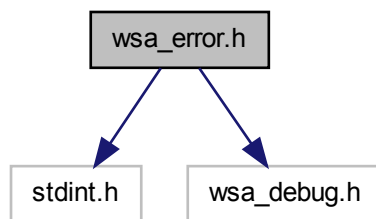
<i>dev</i>	- A pointer to the WSA device structure.
<i>option</i>	- An integer mode of selection: 0 - Off, 1 - On.

**Returns**

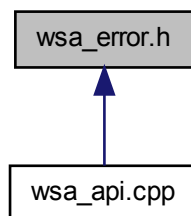
0 on success, or a negative number on error.

**5.4 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_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_PLLOCKFAILED` (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)

#### 5.4.1 Define Documentation

- 5.4.1.1 #define LNEG\_NUM (-10000)
- 5.4.1.2 #define WSA\_ERR\_CMDSENDFAILED (LNEG\_NUM - 1501)
- 5.4.1.3 #define WSA\_ERR\_ETHERNETCONNECTFAILED (LNEG\_NUM - 207)
- 5.4.1.4 #define WSA\_ERR\_ETHERNETINITFAILED (LNEG\_NUM - 209)
- 5.4.1.5 #define WSA\_ERR\_ETHERNETNOTAVBL (LNEG\_NUM - 206)
- 5.4.1.6 #define WSA\_ERR\_FREQOUTOFBOUND (LNEG\_NUM - 601)
- 5.4.1.7 #define WSA\_ERR\_FREQSETFAILED (LNEG\_NUM - 603)
- 5.4.1.8 #define WSA\_ERR\_INITFAILED (LNEG\_NUM - 105)
- 5.4.1.9 #define WSA\_ERR\_INVADCCORRVALUE (LNEG\_NUM - 106)
- 5.4.1.10 #define WSA\_ERR\_INVAMP (LNEG\_NUM - 301)
- 5.4.1.11 #define WSA\_ERR\_INVDWELL (LNEG\_NUM - 1208)
- 5.4.1.12 #define WSA\_ERR\_INVFREQRES (LNEG\_NUM - 602)
- 5.4.1.13 #define WSA\_ERR\_INVGAIN (LNEG\_NUM - 801)
- 5.4.1.14 #define WSA\_ERR\_INVINTFMETHOD (LNEG\_NUM - 201)
- 5.4.1.15 #define WSA\_ERR\_INVIPADDRESS (LNEG\_NUM - 2)
- 5.4.1.16 #define WSA\_ERR\_INVIPHOSTADDRESS (LNEG\_NUM - 202)
- 5.4.1.17 #define WSA\_ERR\_INVNUMBER (LNEG\_NUM - 2000)
- 5.4.1.18 #define WSA\_ERR\_INVNUMFRAMES (LNEG\_NUM - 1209)
- 5.4.1.19 #define WSA\_ERR\_INVREGADDR (LNEG\_NUM - 2001)
- 5.4.1.20 #define WSA\_ERR\_INVRUNMODE (LNEG\_NUM - 1001)
- 5.4.1.21 #define WSA\_ERR\_INVSAMPLESIZE (LNEG\_NUM - 403)
- 5.4.1.22 #define WSA\_ERR\_INVSTARTRES (LNEG\_NUM - 1205)
- 5.4.1.23 #define WSA\_ERR\_INVSTOPFREQ (LNEG\_NUM - 1202)
- 5.4.1.24 #define WSA\_ERR\_INVSTOPRES (LNEG\_NUM - 1206)
- 5.4.1.25 #define WSA\_ERR\_INVTRIGID (LNEG\_NUM - 1201)

5.4.1.26 #define WSA\_ERR\_INVTRIGRANGE (LNEG\_NUM - 1207)

5.4.1.27 #define WSA\_ERR\_MALLOCF FAILED (LNEG\_NUM - 2002)

5.4.1.28 #define WSA\_ERR\_NOCTRLPIPE (LNEG\_NUM - 3)

5.4.1.29 #define WSA\_ERR\_NODATABUS (LNEG\_NUM - 401)

5.4.1.30 #define WSA\_ERR\_NOWSA (LNEG\_NUM - 1)

5.4.1.31 #define WSA\_ERR\_OPENFAILED (LNEG\_NUM - 104)

5.4.1.32 #define WSA\_ERR\_PLLOCKFAILED (LNEG\_NUM - 604)

5.4.1.33 #define WSA\_ERR\_PRODOBSOLETE (LNEG\_NUM - 8)

5.4.1.34 #define WSA\_ERR\_READFRAMEFAILED (LNEG\_NUM - 402)

5.4.1.35 #define WSA\_ERR\_SETFAILED (LNEG\_NUM - 103)

5.4.1.36 #define WSA\_ERR\_STARTOOB (LNEG\_NUM - 1203)

5.4.1.37 #define WSA\_ERR\_STOPOOB (LNEG\_NUM - 1204)

5.4.1.38 #define WSA\_ERR\_UNKNOWN\_ERROR (LNEG\_NUM - 2003)

5.4.1.39 #define WSA\_ERR\_UNKNOWNFWRVSN (LNEG\_NUM - 6)

5.4.1.40 #define WSA\_ERR\_UNKNOWNPRODSE (LNEG\_NUM - 4)

5.4.1.41 #define WSA\_ERR\_UNKNOWNPRODVSN (LNEG\_NUM - 5)

5.4.1.42 #define WSA\_ERR\_UNKNOWNRFEVSN (LNEG\_NUM - 7)

5.4.1.43 #define WSA\_ERR\_USBINITFAILED (LNEG\_NUM - 205)

5.4.1.44 #define WSA\_ERR\_USBNOTAVBL (LNEG\_NUM - 203)

5.4.1.45 #define WSA\_ERR\_USBOPENFAILED (LNEG\_NUM - 204)

5.4.1.46 #define WSA\_ERR\_WSAINUSE (LNEG\_NUM - 102)

5.4.1.47 #define WSA\_ERR\_WSANOTRDY (LNEG\_NUM - 101)

#### 5.4.2 Function Documentation

5.4.2.1 const char\* wsa\_get\_err\_msg ( int16\_t err\_id )

### 5.5 wsa\_lib.txt File Reference

Contain some code documents for wsa\_lib.h.

### 5.5.1 Detailed Description

## Index

- cmd
  - wsa\_socket, 6
- configurations
  - ReadMe.txt, 7
- data
  - wsa\_socket, 6
- descr
  - wsa\_device, 4
- files
  - ReadMe.txt, 7
- freq
  - wsa\_frame\_header, 5
- fw\_version
  - wsa\_descriptor, 3
- gain
  - wsa\_frame\_header, 5
- HIGH
  - wsa\_api.h, 17
- inst\_bw
  - wsa\_descriptor, 3
- intf\_type
  - wsa\_descriptor, 3
- LNEG\_NUM
  - wsa\_error.h, 26
- LOW
  - wsa\_api.h, 17
- max\_sample\_size
  - wsa\_descriptor, 3
- max\_tune\_freq
  - wsa\_descriptor, 3
- MEDIUM
  - wsa\_api.h, 17
- min\_tune\_freq
  - wsa\_descriptor, 3
- nsec
  - wsa\_time, 6
- platforms
  - ReadMe.txt, 7
- prod\_name
  - wsa\_descriptor, 3
- prod\_serial
  - wsa\_descriptor, 3
  - wsa\_frame\_header, 5
- prod\_version
  - wsa\_descriptor, 3
- ReadMe.txt, 7
  - configurations, 7
  - files, 7
  - platforms, 7
- rfe\_name
  - wsa\_descriptor, 3
- rfe\_version
  - wsa\_descriptor, 3
- sample\_size
  - wsa\_frame\_header, 5
- sec
  - wsa\_time, 6
- sock
  - wsa\_device, 4
- time\_stamp
  - wsa\_frame\_header, 5
- ULOW
  - wsa\_api.h, 17
- wsa\_api.h
  - HIGH, 17
  - LOW, 17
  - MEDIUM, 17
  - ULOW, 17
- wsa\_api.cpp, 7
  - wsa\_check\_addr, 8
  - wsa\_check\_cal\_mode, 8
  - wsa\_close, 9
  - wsa\_get\_abs\_max\_amp, 9
  - wsa\_get\_antenna, 9
  - wsa\_get\_bpf, 9
  - wsa\_get\_freq, 10
  - wsa\_get\_gain, 10
  - wsa\_get\_lpf, 10
  - wsa\_is\_connected, 10
  - wsa\_list, 10
  - wsa\_open, 11
  - wsa\_read\_pkt, 12

- wsa\_run\_cal\_mode, [12](#)
- wsa\_set\_antenna, [13](#)
- wsa\_set\_bpf, [13](#)
- wsa\_set\_freq, [13](#)
- wsa\_set\_gain, [14](#)
- wsa\_set\_lpf, [14](#)
- wsa\_verify\_freq, [14](#)
- wsa\_api.h, [15](#)
- wsa\_check\_addr, [17](#)
- wsa\_check\_cal\_mode, [18](#)
- wsa\_close, [18](#)
- wsa\_gain, [17](#)
- wsa\_get\_abs\_max\_amp, [18](#)
- wsa\_get\_antenna, [19](#)
- wsa\_get\_bpf, [19](#)
- wsa\_get\_freq, [19](#)
- wsa\_get\_gain, [19](#)
- wsa\_get\_lpf, [19](#)
- wsa\_is\_connected, [20](#)
- wsa\_list, [20](#)
- wsa\_open, [20](#)
- wsa\_read\_pkt, [21](#)
- wsa\_run\_cal\_mode, [21](#)
- wsa\_set\_antenna, [22](#)
- wsa\_set\_bpf, [22](#)
- wsa\_set\_freq, [22](#)
- wsa\_set\_gain, [23](#)
- wsa\_set\_lpf, [23](#)
- wsa\_check\_addr
  - wsa\_api.cpp, [8](#)
  - wsa\_api.h, [17](#)
- wsa\_check\_cal\_mode
  - wsa\_api.cpp, [8](#)
  - wsa\_api.h, [18](#)
- wsa\_close
  - wsa\_api.cpp, [9](#)
  - wsa\_api.h, [18](#)
- wsa\_descriptor, [2](#)
  - fw\_version, [3](#)
  - inst\_bw, [3](#)
  - intf\_type, [3](#)
  - max\_sample\_size, [3](#)
  - max\_tune\_freq, [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\_ERR\_CMDSENDFailed
  - wsa\_error.h, [26](#)
- WSA\_ERR\_ETHERNETCONNECTFailed
  - wsa\_error.h, [26](#)
- WSA\_ERR\_ETHERNETINITFailed
  - wsa\_error.h, [26](#)
- WSA\_ERR\_ETHERNETNOTAVBL
  - wsa\_error.h, [26](#)
- WSA\_ERR\_FREQOUTOFBOUND
  - wsa\_error.h, [26](#)
- WSA\_ERR\_FREQSETFailed
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INITFailed
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVADCCORRVALUE
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVAMP
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVDWELL
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVFREQRES
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVGAIN
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVINTFMETHOD
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVIPADDRESS
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVIPHOSTADDRESS
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVNUMBER
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVNUMFRAMES
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVREGADDR
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVRUNMODE
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVSAMPLESIZE
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVSTARTRES
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVSTOPFREQ
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVSTOPRES
  - wsa\_error.h, [26](#)
- WSA\_ERR\_INVTRIGID
  - wsa\_error.h, [26](#)



- WSA\_ERR\_INVTRIGRANGE
  - wsa\_error.h, [26](#)
- WSA\_ERR\_MALLOCF FAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_NOCTRLPIPE
  - wsa\_error.h, [27](#)
- WSA\_ERR\_NODATABUS
  - wsa\_error.h, [27](#)
- WSA\_ERR\_NOWSA
  - wsa\_error.h, [27](#)
- WSA\_ERR\_OPENFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_PLLOCKFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_PRODOBSOLETE
  - wsa\_error.h, [27](#)
- WSA\_ERR\_READFRAMEFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_SETFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_STARTOOB
  - wsa\_error.h, [27](#)
- WSA\_ERR\_STOPOOB
  - wsa\_error.h, [27](#)
- WSA\_ERR\_UNKNOWN\_ERROR
  - wsa\_error.h, [27](#)
- WSA\_ERR\_UNKNOWNFWRVSN
  - wsa\_error.h, [27](#)
- WSA\_ERR\_UNKNOWNPRODSE
  - wsa\_error.h, [27](#)
- WSA\_ERR\_UNKNOWNPRODVSN
  - wsa\_error.h, [27](#)
- WSA\_ERR\_UNKNOWNRFEVSN
  - wsa\_error.h, [27](#)
- WSA\_ERR\_USBINITFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_USBNOTAVBL
  - wsa\_error.h, [27](#)
- WSA\_ERR\_USBOPENFAILED
  - wsa\_error.h, [27](#)
- WSA\_ERR\_WSAINUSE
  - wsa\_error.h, [27](#)
- WSA\_ERR\_WSANOTRDY
  - wsa\_error.h, [27](#)
- wsa\_error.h, [24](#)
  - LNEG\_NUM, [26](#)
  - WSA\_ERR\_CMDSENDFAILED, [26](#)
  - WSA\_ERR\_ETHERNETCONNECTFAILED, freq, [5](#)
  - [26](#)
- WSA\_ERR\_ETHERNETINITFAILED, [26](#)
- WSA\_ERR\_ETHERNETNOTAVBL, [26](#)
- WSA\_ERR\_FREQOUTOFBOUND, [26](#)
- WSA\_ERR\_FREQSETFAILED, [26](#)
- WSA\_ERR\_INITFAILED, [26](#)
- WSA\_ERR\_INVADCCORRVALUE, [26](#)
- WSA\_ERR\_INVAMP, [26](#)
- WSA\_ERR\_INVDWELL, [26](#)
- WSA\_ERR\_INVFREQRES, [26](#)
- WSA\_ERR\_INVGAIN, [26](#)
- WSA\_ERR\_INVINTFMETHOD, [26](#)
- WSA\_ERR\_INVIPADDRESS, [26](#)
- WSA\_ERR\_INVIPHOSTADDRESS, [26](#)
- WSA\_ERR\_INVNUMBER, [26](#)
- WSA\_ERR\_INVNUMFRAMES, [26](#)
- WSA\_ERR\_INVREGADDR, [26](#)
- WSA\_ERR\_INVRUNMODE, [26](#)
- WSA\_ERR\_INVSAMPLESIZE, [26](#)
- WSA\_ERR\_INVSTARTRES, [26](#)
- WSA\_ERR\_INVSTOPFREQ, [26](#)
- WSA\_ERR\_INVSTOPRES, [26](#)
- WSA\_ERR\_INVTRIGID, [26](#)
- WSA\_ERR\_INVTRIGRANGE, [26](#)
- WSA\_ERR\_MALLOCF FAILED, [27](#)
- WSA\_ERR\_NOCTRLPIPE, [27](#)
- WSA\_ERR\_NODATABUS, [27](#)
- WSA\_ERR\_NOWSA, [27](#)
- WSA\_ERR\_OPENFAILED, [27](#)
- WSA\_ERR\_PLLOCKFAILED, [27](#)
- WSA\_ERR\_PRODOBSOLETE, [27](#)
- WSA\_ERR\_READFRAMEFAILED, [27](#)
- WSA\_ERR\_SETFAILED, [27](#)
- WSA\_ERR\_STARTOOB, [27](#)
- WSA\_ERR\_STOPOOB, [27](#)
- WSA\_ERR\_UNKNOWN\_ERROR, [27](#)
- WSA\_ERR\_UNKNOWNFWRVSN, [27](#)
- WSA\_ERR\_UNKNOWNPRODSE, [27](#)
- WSA\_ERR\_UNKNOWNPRODVSN, [27](#)
- WSA\_ERR\_UNKNOWNRFEVSN, [27](#)
- WSA\_ERR\_USBINITFAILED, [27](#)
- WSA\_ERR\_USBNOTAVBL, [27](#)
- WSA\_ERR\_USBOPENFAILED, [27](#)
- WSA\_ERR\_WSAINUSE, [27](#)
- WSA\_ERR\_WSANOTRDY, [27](#)
- wsa\_get\_err\_msg, [27](#)
- wsa\_frame\_header, [4](#)
- gain, [5](#)
- prod\_serial, [5](#)

- sample\_size, 5
- time\_stamp, 5
- wsa\_gain
  - wsa\_api.h, 17
- wsa\_get\_abs\_max\_amp
  - wsa\_api.cpp, 9
  - wsa\_api.h, 18
- wsa\_get\_antenna
  - wsa\_api.cpp, 9
  - wsa\_api.h, 19
- wsa\_get\_bpf
  - wsa\_api.cpp, 9
  - wsa\_api.h, 19
- wsa\_get\_err\_msg
  - wsa\_error.h, 27
- wsa\_get\_freq
  - wsa\_api.cpp, 10
  - wsa\_api.h, 19
- wsa\_get\_gain
  - wsa\_api.cpp, 10
  - wsa\_api.h, 19
- wsa\_get\_lpf
  - wsa\_api.cpp, 10
  - wsa\_api.h, 19
- wsa\_is\_connected
  - wsa\_api.cpp, 10
  - wsa\_api.h, 20
- wsa\_lib.txt, 27
- wsa\_list
  - wsa\_api.cpp, 10
  - wsa\_api.h, 20
- wsa\_open
  - wsa\_api.cpp, 11
  - wsa\_api.h, 20
- wsa\_read\_pkt
  - wsa\_api.cpp, 12
  - wsa\_api.h, 21
- wsa\_resp, 6
- wsa\_run\_cal\_mode
  - wsa\_api.cpp, 12
  - wsa\_api.h, 21
- wsa\_set\_antenna
  - wsa\_api.cpp, 13
  - wsa\_api.h, 22
- wsa\_set\_bpf
  - wsa\_api.cpp, 13
  - wsa\_api.h, 22
- wsa\_set\_freq
  - wsa\_api.cpp, 13
  - wsa\_api.h, 22
- wsa\_set\_gain
  - wsa\_api.cpp, 14
  - wsa\_api.h, 23
- wsa\_set\_lpf
  - wsa\_api.cpp, 14
  - wsa\_api.h, 23
- wsa\_socket, 6
  - cmd, 6
  - data, 6
- wsa\_time, 6
  - nsec, 6
  - sec, 6
- wsa\_verify\_freq
  - wsa\_api.cpp, 14