

Standard WSA Library

Generated by Doxygen 1.7.4

Mon Aug 29 2011 16:10:42

Contents

1	Introduction	1
1.1	How to use the library	1
2	Data Structure Index	1
2.1	Data Structures	1
3	File Index	1
3.1	File List	2
4	Data Structure Documentation	2
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	5
4.5.1	Field Documentation	6
4.6	wsa_time Struct Reference	6
4.6.1	Field Documentation	6
5	File Documentation	6
5.1	ReadMe.txt File Reference	6
5.1.1	Variable Documentation	7
5.2	wsa_error.h File Reference	7
5.2.1	Define Documentation	9
5.2.2	Function Documentation	10
5.3	wsa_lib.cpp File Reference	11
5.3.1	Function Documentation	11
5.4	wsa_lib.h File Reference	15
5.4.1	Define Documentation	17
5.4.2	Enumeration Type Documentation	17

5.4.3	Function Documentation	17
5.5	wsa_lib.txt File Reference	20
5.5.1	Detailed Description	20

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:

wsa_descriptor (This structure stores WSA information)	2
wsa_device (A structure containing the components associate with each WSA device)	3
wsa_frame_header (This structure contains header information related to each frame read by wsa_get_frame())	4
wsa_resp (This structure contains the response information for each query)	5
wsa_socket (A structure containing the socket parameters used for creating TCP/IP connection for control and data acquisition)	5
wsa_time (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:

wsa_error.h	7
wsa_lib.cpp	11
wsa_lib.h	15

4 Data Structure Documentation

4.1 wsa_descriptor Struct Reference

This structure stores WSA information.

Data Fields

- char [prod_name](#) [50]
- char [prod_serial](#) [20]
- char [prod_version](#) [20]
- char [rfe_name](#) [50]
- char [rfe_version](#) [20]
- char [fw_version](#) [20]
- char [intf_type](#) [20]
- uint64_t [inst_bw](#)
- uint64_t [max_sample_size](#)
- uint64_t [max_tune_freq](#)
- uint64_t [min_tune_freq](#)

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.

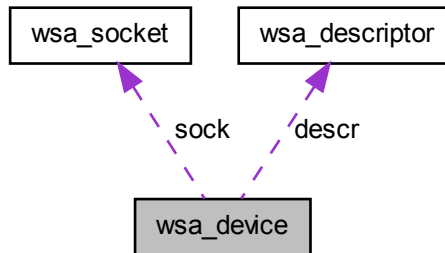
4.1.1.11 char rfe_version

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](#).

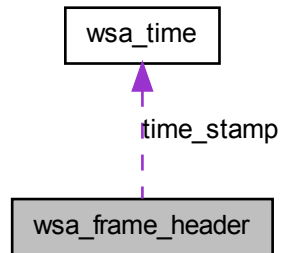
4.2.1.2 struct `wsa_socket` `sock`

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.

Data Fields

- int64_t [status](#)
- char * [result](#)

4.4.1 Field Documentation

4.4.1.1 char result

The resulted string responded to a query.

4.4.1.2 int32_t status

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

4.5 wsa_socket Struct Reference

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

Data Fields

- SOCKET [cmd](#)
- SOCKET [data](#)

4.5.1 Field Documentation

4.5.1.1 SOCKET cmd

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

4.5.1.2 SOCKET data

The data socket used for streaming of data

4.6 wsa_time Struct Reference

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

Data Fields

- `int32_t` [sec](#)
- `uint32_t` [nsec](#)

4.6.1 Field Documentation

4.6.1.1 `int32_t` [nsec](#)

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

4.6.1.2 `int32_t` [sec](#)

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

5 File Documentation

5.1 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

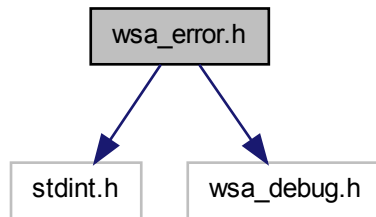
5.1.1.1 and information about the [configurations](#)

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](#)

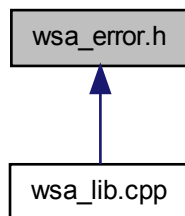
5.1.1.3 and information about the [platforms](#)

5.2 wsa_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_PLLLOCKFAILED` (LNEG_NUM - 604)
- `#define WSA_ERR_INVGAIN` (LNEG_NUM - 801)
- `#define WSA_ERR_INVRUNMODE` (LNEG_NUM - 1001)
- `#define WSA_ERR_INVTRIGID` (LNEG_NUM - 1201)
- `#define WSA_ERR_INVSTOPFREQ` (LNEG_NUM - 1202)
- `#define WSA_ERR_STARTOOB` (LNEG_NUM - 1203)
- `#define WSA_ERR_STOPOOB` (LNEG_NUM - 1204)
- `#define WSA_ERR_INVSTARTRES` (LNEG_NUM - 1205)
- `#define WSA_ERR_INVSTOPRES` (LNEG_NUM - 1206)
- `#define WSA_ERR_INVTRIGRANGE` (LNEG_NUM - 1207)
- `#define WSA_ERR_INVDWELL` (LNEG_NUM - 1208)
- `#define WSA_ERR_INVNUMFRAMES` (LNEG_NUM - 1209)
- `#define WSA_ERR_CMDSENDFAILED` (LNEG_NUM - 1501)
- `#define WSA_ERR_INVNUMBER` (LNEG_NUM - 2000)
- `#define WSA_ERR_INVREGADDR` (LNEG_NUM - 2001)
- `#define WSA_ERR_MALLOCFAILED` (LNEG_NUM - 2002)
- `#define WSA_ERR_UNKNOWN_ERROR` (LNEG_NUM - 2003)

Functions

- `const char * wsa_get_err_msg` (int16_t err_id)

5.2.1 Define Documentation

- 5.2.1.1 #define LNEG_NUM (-10000)
- 5.2.1.2 #define WSA_ERR_CMDSENDFAILED (LNEG_NUM - 1501)
- 5.2.1.3 #define WSA_ERR_ETHERNETCONNECTFAILED (LNEG_NUM - 207)
- 5.2.1.4 #define WSA_ERR_ETHERNETINITFAILED (LNEG_NUM - 209)
- 5.2.1.5 #define WSA_ERR_ETHERNETNOTAVBL (LNEG_NUM - 206)
- 5.2.1.6 #define WSA_ERR_FREQOUTOFBOUND (LNEG_NUM - 601)
- 5.2.1.7 #define WSA_ERR_FREQSETFAILED (LNEG_NUM - 603)
- 5.2.1.8 #define WSA_ERR_INITFAILED (LNEG_NUM - 105)
- 5.2.1.9 #define WSA_ERR_INVADCCORRVALUE (LNEG_NUM - 106)
- 5.2.1.10 #define WSA_ERR_INVAMP (LNEG_NUM - 301)
- 5.2.1.11 #define WSA_ERR_INVDWELL (LNEG_NUM - 1208)
- 5.2.1.12 #define WSA_ERR_INVFREQRES (LNEG_NUM - 602)
- 5.2.1.13 #define WSA_ERR_INVGAIN (LNEG_NUM - 801)
- 5.2.1.14 #define WSA_ERR_INVINTFMETHOD (LNEG_NUM - 201)
- 5.2.1.15 #define WSA_ERR_INVIPADDRESS (LNEG_NUM - 2)
- 5.2.1.16 #define WSA_ERR_INVIPHOSTADDRESS (LNEG_NUM - 202)
- 5.2.1.17 #define WSA_ERR_INVNUMBER (LNEG_NUM - 2000)
- 5.2.1.18 #define WSA_ERR_INVNUMFRAMES (LNEG_NUM - 1209)
- 5.2.1.19 #define WSA_ERR_INVREGADDR (LNEG_NUM - 2001)
- 5.2.1.20 #define WSA_ERR_INVRUNMODE (LNEG_NUM - 1001)
- 5.2.1.21 #define WSA_ERR_INVSAMPLESIZE (LNEG_NUM - 403)
- 5.2.1.22 #define WSA_ERR_INVSTARTRES (LNEG_NUM - 1205)
- 5.2.1.23 #define WSA_ERR_INVSTOPFREQ (LNEG_NUM - 1202)
- 5.2.1.24 #define WSA_ERR_INVSTOPRES (LNEG_NUM - 1206)
- 5.2.1.25 #define WSA_ERR_INVTRIGID (LNEG_NUM - 1201)

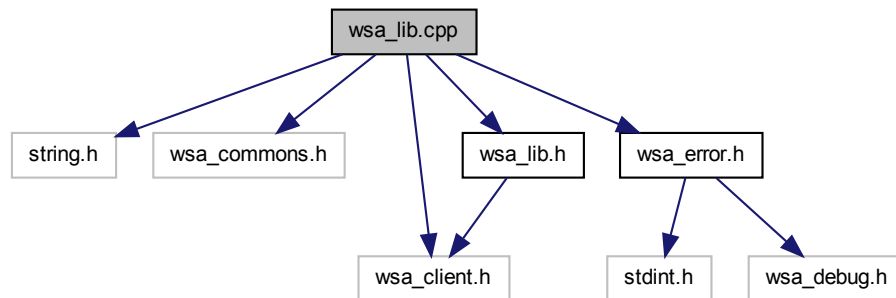
- 5.2.1.26 `#define WSA_ERR_INVTRIGRANGE (LNEG_NUM - 1207)`
- 5.2.1.27 `#define WSA_ERR_MALLOCF FAILED (LNEG_NUM - 2002)`
- 5.2.1.28 `#define WSA_ERR_NOCTRLPIPE (LNEG_NUM - 3)`
- 5.2.1.29 `#define WSA_ERR_NODATABUS (LNEG_NUM - 401)`
- 5.2.1.30 `#define WSA_ERR_NOWSA (LNEG_NUM - 1)`
- 5.2.1.31 `#define WSA_ERR_OPENFAILED (LNEG_NUM - 104)`
- 5.2.1.32 `#define WSA_ERR_PLLOCKFAILED (LNEG_NUM - 604)`
- 5.2.1.33 `#define WSA_ERR_PRODOBSOLETE (LNEG_NUM - 8)`
- 5.2.1.34 `#define WSA_ERR_READFRAMEFAILED (LNEG_NUM - 402)`
- 5.2.1.35 `#define WSA_ERR_SETFAILED (LNEG_NUM - 103)`
- 5.2.1.36 `#define WSA_ERR_STARTOOB (LNEG_NUM - 1203)`
- 5.2.1.37 `#define WSA_ERR_STOPOOB (LNEG_NUM - 1204)`
- 5.2.1.38 `#define WSA_ERR_UNKNOWN_ERROR (LNEG_NUM - 2003)`
- 5.2.1.39 `#define WSA_ERR_UNKNOWNFWRVSN (LNEG_NUM - 6)`
- 5.2.1.40 `#define WSA_ERR_UNKNOWNPRODSE (LNEG_NUM - 4)`
- 5.2.1.41 `#define WSA_ERR_UNKNOWNPRODVSN (LNEG_NUM - 5)`
- 5.2.1.42 `#define WSA_ERR_UNKNOWNRFEVSN (LNEG_NUM - 7)`
- 5.2.1.43 `#define WSA_ERR_USBINITFAILED (LNEG_NUM - 205)`
- 5.2.1.44 `#define WSA_ERR_USBNOTAVBL (LNEG_NUM - 203)`
- 5.2.1.45 `#define WSA_ERR_USBOPENFAILED (LNEG_NUM - 204)`
- 5.2.1.46 `#define WSA_ERR_WSAINUSE (LNEG_NUM - 102)`
- 5.2.1.47 `#define WSA_ERR_WSANOTRDY (LNEG_NUM - 101)`

5.2.2 Function Documentation

- 5.2.2.1 `const char* wsa_get_err_msg (int16_t err_id)`

5.3 wsa_lib.cpp File Reference

Include dependency graph for wsa_lib.cpp:



Functions

- `int16_t wsa_dev_init` (struct `wsa_device` *dev)
- `int16_t wsa_connect` (struct `wsa_device` *dev, char *cmd_syntax, char *intf_method)
- `int16_t wsa_disconnect` (struct `wsa_device` *dev)
- `int16_t wsa_list_devs` (char **wsa_list)
- `int16_t wsa_send_command` (struct `wsa_device` *dev, char *command)
- `struct wsa_resp wsa_send_query` (struct `wsa_device` *dev, char *command)
- `int16_t wsa_query_error` (struct `wsa_device` *dev)
- `int64_t wsa_get_frame` (struct `wsa_device` *dev, struct `wsa_frame_header` *header, int32_t *i_buf, int32_t *q_buf, uint64_t sample_size)

5.3.1 Function Documentation

5.3.1.1 `int16_t wsa_connect (struct wsa_device * dev, char * cmd_syntax, char * intf_method)`

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

Parameters

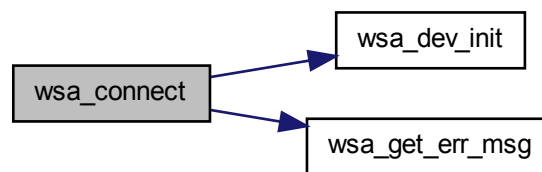
<i>dev</i>	- A pointer to the WSA device structure to be connected/established.
<i>cmd_syntax</i>	- A char pointer to store standard for control commands communication to the WSA. Currently supported standard command syntax type is: SCPI.

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

Returns

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

Here is the call graph for this function:



5.3.1.2 `int16_t wsa_dev_init (struct wsa_device * dev)`

Initialized the the [wsa_device](#) structure

Parameters

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

Returns

None

5.3.1.3 `int16_t wsa_disconnect (struct wsa_device * dev)`

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

Parameters

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

Returns

0 on success, or a negative number on error.

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

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

Parameters

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

Returns

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

5.3.1.5 `int16_t wsa_list_devs (char ** wsa_list)`

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

Parameters

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

Returns

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

5.3.1.6 `int16_t wsa_query_error (struct wsa_device * dev)`

Query the WSA for any error.

Parameters

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

Returns

0 on success, or a negative number on error.

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

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

Parameters

<i>dev</i>	- The WSA device structure from which the help information will be provided.
------------	--

Returns

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

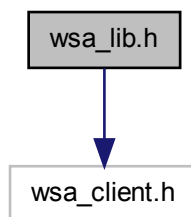
The result stored in a [wsa_resp](#) struct format.

Here is the call graph for this function:

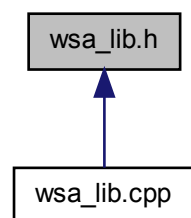


5.4 wsa_lib.h File Reference

Include dependency graph for `wsa_lib.h`:



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



Data Structures

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

Defines

- #define [FALSE](#) 0
- #define [TRUE](#) 1
- #define [SCPI](#) "SCPI"

Enumerations

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

Functions

- int16_t [wsa_connect](#) (struct [wsa_device](#) *dev, char *cmd_syntax, char *intf_method)
- int16_t [wsa_disconnect](#) (struct [wsa_device](#) *dev)
- int16_t [wsa_list_devs](#) (char **wsa_list)
- int16_t [wsa_send_command](#) (struct [wsa_device](#) *dev, char *command)
- struct [wsa_resp](#) [wsa_send_query](#) (struct [wsa_device](#) *dev, char *command)
- int16_t [wsa_query_error](#) (struct [wsa_device](#) *dev)
- int64_t [wsa_get_frame](#) (struct [wsa_device](#) *dev, struct [wsa_frame_header](#) *header, int32_t *i_buf, int32_t *q_buf, uint64_t sample_size)

5.4.1 Define Documentation

5.4.1.1 `#define FALSE 0`

5.4.1.2 `#define SCPI "SCPI"`

5.4.1.3 `#define TRUE 1`

5.4.2 Enumeration Type Documentation

5.4.2.1 `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.

5.4.3 Function Documentation

5.4.3.1 `int16_t wsa_connect (struct wsa_device * dev, char * cmd_syntax, char * intf_method)`

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

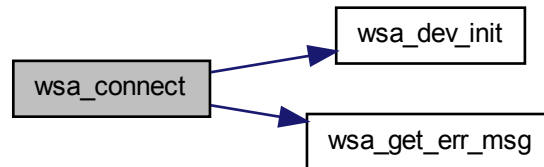
Parameters

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

Returns

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

Here is the call graph for this function:



5.4.3.2 `int16_t wsa_disconnect (struct wsa_device * dev)`

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

Parameters

<code>dev</code>	- A pointer to the WSA device structure to be closed.
------------------	---

Returns

0 on success, or a negative number on error.

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

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

Parameters

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

Returns

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

5.4.3.4 `int16_t wsa_list_devs (char ** wsa_list)`

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

Parameters

<code>wsa_list</code>	- 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.

5.4.3.5 `int16_t wsa_query_error (struct wsa_device * dev)`

Query the WSA for any error.

Parameters

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

Returns

0 on success, or a negative number on error.

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

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

Parameters

<code>dev</code>	- The WSA device structure from which the help information will be provided.
------------------	--

Returns

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

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

Here is the call graph for this function:



5.5 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
- FALSE
 - wsa_lib.h, 17
- files
 - ReadMe.txt, 7
- freq
 - wsa_frame_header, 5
- fw_version
 - wsa_descriptor, 2
- gain
 - wsa_frame_header, 5
- HIGH
 - wsa_lib.h, 17
- inst_bw
 - wsa_descriptor, 2
- intf_type
 - wsa_descriptor, 2
- LNEG_NUM
 - wsa_error.h, 9
- LOW
 - wsa_lib.h, 17
- max_sample_size
 - wsa_descriptor, 2
- max_tune_freq
 - wsa_descriptor, 2
- MEDIUM
 - wsa_lib.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, 6
 - configurations, 7
 - files, 7
 - platforms, 7
- result
 - wsa_resp, 5
- rfe_name
 - wsa_descriptor, 3
- rfe_version
 - wsa_descriptor, 3
- sample_size
 - wsa_frame_header, 5
- SCPI
 - wsa_lib.h, 17
- sec
 - wsa_time, 6
- sock
 - wsa_device, 4
- status
 - wsa_resp, 5
- time_stamp
 - wsa_frame_header, 5
- TRUE
 - wsa_lib.h, 17
- ULOW
 - wsa_lib.h, 17
- wsa_lib.h
 - HIGH, 17
 - LOW, 17
 - MEDIUM, 17
 - ULOW, 17
- wsa_connect
 - wsa_lib.cpp, 11
 - wsa_lib.h, 17
- wsa_descriptor, 2

- fw_version, [2](#)
- inst_bw, [2](#)
- intf_type, [2](#)
- max_sample_size, [2](#)
- max_tune_freq, [2](#)
- min_tune_freq, [3](#)
- prod_name, [3](#)
- prod_serial, [3](#)
- prod_version, [3](#)
- rfe_name, [3](#)
- rfe_version, [3](#)
- wsa_dev_init
 - wsa_lib.cpp, [12](#)
- wsa_device, [3](#)
 - descr, [4](#)
 - sock, [4](#)
- wsa_disconnect
 - wsa_lib.cpp, [12](#)
 - wsa_lib.h, [18](#)
- 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_FREQOUTOFBOUND
 - wsa_error.h, [9](#)
- WSA_ERR_FREQSETFAILED
 - 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_INVDWELL
 - wsa_error.h, [9](#)
- WSA_ERR_INVFREQRES
 - wsa_error.h, [9](#)
- WSA_ERR_INVGAIN
 - wsa_error.h, [9](#)
- WSA_ERR_INVINTFMETHOD
 - wsa_error.h, [9](#)
- WSA_ERR_INVIPADDRESS
 - wsa_error.h, [9](#)
- WSA_ERR_INVIPHOSTADDRESS
 - wsa_error.h, [9](#)
- WSA_ERR_INVNUMBER
 - wsa_error.h, [9](#)
- WSA_ERR_INVNUMFRAMES
 - wsa_error.h, [9](#)
- WSA_ERR_INVREGADDR
 - wsa_error.h, [9](#)
- WSA_ERR_INVRUNMODE
 - wsa_error.h, [9](#)
- WSA_ERR_INVSAMPLESIZE
 - wsa_error.h, [9](#)
- WSA_ERR_INVSTARTRES
 - wsa_error.h, [9](#)
- WSA_ERR_INVSTOPFREQ
 - wsa_error.h, [9](#)
- WSA_ERR_INVSTOPRES
 - wsa_error.h, [9](#)
- 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_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, [10](#)
- WSA_ERR_READFRAMEFAILED
 - wsa_error.h, [10](#)
- WSA_ERR_SETFAILED
 - wsa_error.h, [10](#)
- WSA_ERR_STARTOOB
 - wsa_error.h, [10](#)
- WSA_ERR_STOPOOB
 - wsa_error.h, [10](#)
- WSA_ERR_UNKNOWN_ERROR
 - wsa_error.h, [10](#)
- WSA_ERR_UNKNOWNFWRVSN
 - wsa_error.h, [10](#)
- WSA_ERR_UNKNOWNPRODSEI
 - wsa_error.h, [10](#)
- WSA_ERR_UNKNOWNPRODVSN
 - wsa_error.h, [10](#)
- WSA_ERR_UNKNOWNRFEVSN

- wsa_error.h, 10
- WSA_ERR_USBINITFAILED
 - wsa_error.h, 10
- WSA_ERR_USBNOTAVBL
 - wsa_error.h, 10
- WSA_ERR_USBOPENFAILED
 - wsa_error.h, 10
- WSA_ERR_WSAINUSE
 - wsa_error.h, 10
- WSA_ERR_WSANOTRDY
 - wsa_error.h, 10
- wsa_error.h, 7
 - LNEG_NUM, 9
 - WSA_ERR_CMDSENDFailed, 9
 - WSA_ERR_ETHERNETCONNECTFAILED, gain, 5
 - 9
 - WSA_ERR_ETHERNETINITFAILED, 9
 - 9
 - WSA_ERR_ETHERNETNOTAVBL, 9
 - WSA_ERR_FREQOUTOFBOUND, 9
 - WSA_ERR_FREQSETFAILED, 9
 - WSA_ERR_INITFAILED, 9
 - WSA_ERR_INVADCCORRVALUE, 9
 - WSA_ERR_INVAMP, 9
 - WSA_ERR_INVDWELL, 9
 - WSA_ERR_INVFREQRES, 9
 - WSA_ERR_INVGAIN, 9
 - WSA_ERR_INVINTFMETHOD, 9
 - WSA_ERR_INVIPADDRESS, 9
 - WSA_ERR_INVIPHOSTADDRESS, 9
 - WSA_ERR_INVNUMBER, 9
 - WSA_ERR_INVNUMFRAMES, 9
 - WSA_ERR_INVREGADDR, 9
 - WSA_ERR_INVRUNMODE, 9
 - WSA_ERR_INVSAMPLESIZE, 9
 - WSA_ERR_INVSTARTRES, 9
 - WSA_ERR_INVSTOPFREQ, 9
 - WSA_ERR_INVSTOPRES, 9
 - WSA_ERR_INVTRIGID, 10
 - WSA_ERR_INVTRIGRANGE, 10
 - WSA_ERR_MALLOCFAILED, 10
 - WSA_ERR_NOCTRLPIPE, 10
 - WSA_ERR_NODATABUS, 10
 - WSA_ERR_NOWSA, 10
 - WSA_ERR_OPENFAILED, 10
 - WSA_ERR_PLLOCKFAILED, 10
 - WSA_ERR_PRODOBSOLETE, 10
 - WSA_ERR_READFRAMEFAILED, 10
 - WSA_ERR_SETFAILED, 10
 - WSA_ERR_STARTOOB, 10
 - WSA_ERR_STOPOOB, 10
 - WSA_ERR_UNKNOWN_ERROR, 10
 - WSA_ERR_UNKNOWNFWRVSN, 10
 - WSA_ERR_UNKNOWNPRODSE, 10
 - WSA_ERR_UNKNOWNPRODVSN, 10
 - WSA_ERR_UNKNOWNRFEVSN, 10
 - WSA_ERR_USBINITFAILED, 10
 - WSA_ERR_USBNOTAVBL, 10
 - WSA_ERR_USBOPENFAILED, 10
 - WSA_ERR_WSAINUSE, 10
 - WSA_ERR_WSANOTRDY, 10
 - wsa_get_err_msg, 10
- wsa_frame_header, 4
 - freq, 5
 - gain, 5
 - prod_serial, 5
 - sample_size, 5
 - time_stamp, 5
- wsa_gain
 - wsa_lib.h, 17
- wsa_get_err_msg
 - wsa_error.h, 10
- wsa_get_frame
 - wsa_lib.cpp, 13
 - wsa_lib.h, 18
- wsa_lib.cpp, 11
 - wsa_connect, 11
 - wsa_dev_init, 12
 - wsa_disconnect, 12
 - wsa_get_frame, 13
 - wsa_list_devs, 13
 - wsa_query_error, 13
 - wsa_send_command, 14
 - wsa_send_query, 14
- wsa_lib.h, 15
 - FALSE, 17
 - SCPI, 17
 - TRUE, 17
 - wsa_connect, 17
 - wsa_disconnect, 18
 - wsa_gain, 17
 - wsa_get_frame, 18
 - wsa_list_devs, 19
 - wsa_query_error, 19
 - wsa_send_command, 19
 - wsa_send_query, 20
- wsa_lib.txt, 20
- wsa_list_devs
 - wsa_lib.cpp, 13
 - wsa_lib.h, 19

- wsa_query_error
 - wsa_lib.cpp, [13](#)
 - wsa_lib.h, [19](#)
- wsa_resp, [5](#)
 - result, [5](#)
 - status, [5](#)
- wsa_send_command
 - wsa_lib.cpp, [14](#)
 - wsa_lib.h, [19](#)
- wsa_send_query
 - wsa_lib.cpp, [14](#)
 - wsa_lib.h, [20](#)
- wsa_socket, [5](#)
 - cmd, [6](#)
 - data, [6](#)
- wsa_time, [6](#)
 - nsec, [6](#)
 - sec, [6](#)