Quantenna Configuration and Status APIs INTERNAL ONLY

Generated by Doxygen 1.8.17

Chapter 1

Overview

This document describes the Quantenna Configuration and Status API (QCSAPI).

The document contains a general overview, specific information on the SDK, details on the RPC mechanism used for remote control, and detailed API usage and data structures used in QCSAPI. In addition to this, a sample application, `call_qcsapi` is detailed.

Chapter 2

Revision history

Chapter 3

References

- Quantenna Software Architecture (Software Integration Guide)
- TR-098: Internet Gateway Device Data Model for TR-069
  Issue: 1 Amendment 2
  Issue Date: September 2008

Chapter 4

Background

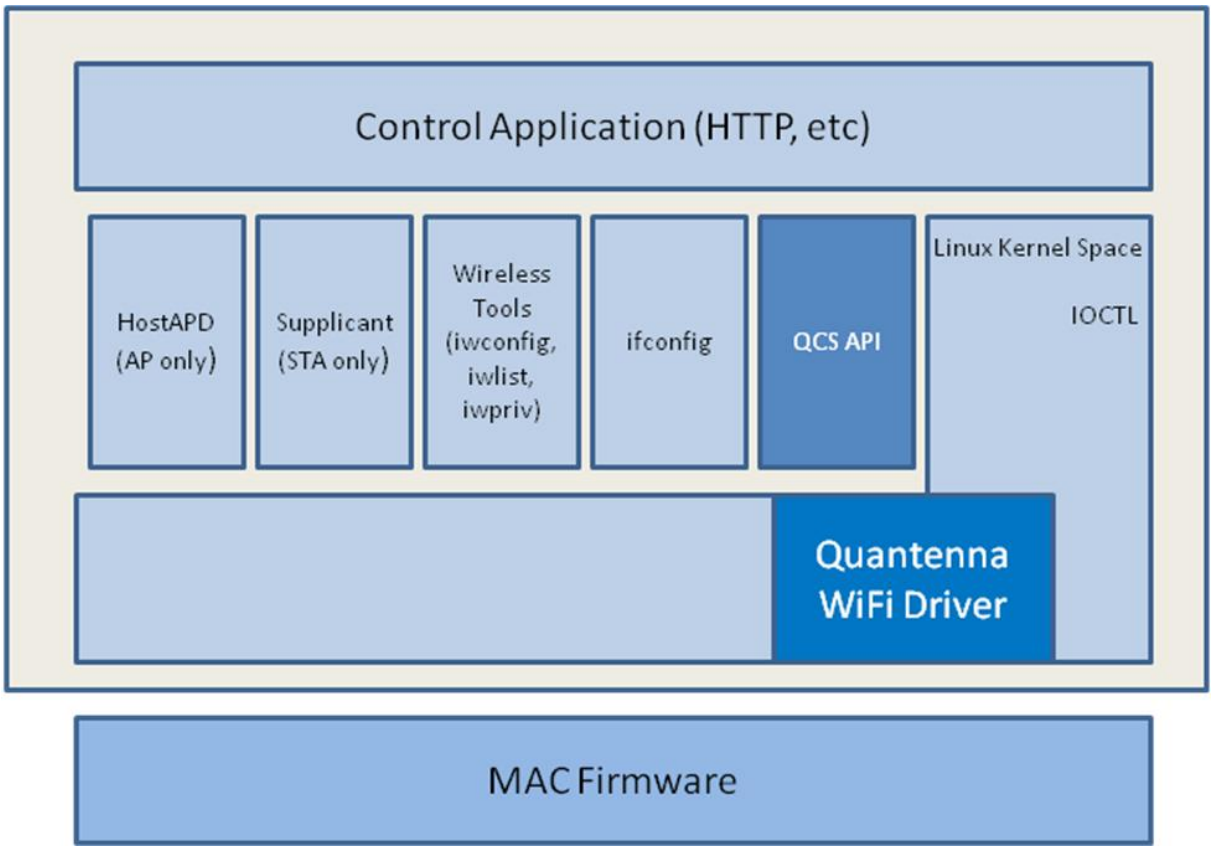
The Quantenna WiFi device includes a number of configuration parameters, e.g. the MAC address or the current WiFi channel. In addition a variety of other parameters can be monitored, including counters and status.

Prior to introducing the QCSAPI, reviewing the current configuration, making changes and monitoring the system required accessing several user-level commands, including ifconfig, iwconfig, iwpriv and iwlist.

While workable for casual access, for more systematic programmable access, some drawbacks are present. First, required information is distributed over several commands or interfaces; no single point of access is available. Second, accessing the configuration is heavy, for the desired command must be executed as a shell process, with the output typically then parsed using a shell script or an awk program. A second child process is thus usually required to interpret the output from the original command. And any changes to the underlying configuration or monitoring command would likely require changes in this interface programming.

The Quantenna Configuration and Status APIs (QCSAPI) address these drawbacks.
They offer the application programmer a series of C-callable APIs to access and modify the current configuration. All configuration parameters can be read and updated through this API. Statistics and other monitoring parameters are also available. And the APIs will not change, regardless of changes to the underlying programming, including the user-level interface commands.

The figure below shows selected software components on the Quantenna WiFI device and their interrelationships:


Control Application (HTTP, etc)
HostAPD (AP only)
Supplicant (STA only)
Wireless Tools (iwconfig, iwlist, iwpriv)
ifconfig
QCS API
Linux Kernel Space
IOCTL
Quantenna WiFi Driver
MAC Firmware

Chapter 5

QCSAPI concepts

5.1 Overview of the QCSAPIs

The QCSAPI set shares a common set of parameters as described below.

5.1.1 Names of the QCSAPIs

The name of each QCSAPI starts with "qcsapi". The underscore character ('_') serves as a separator and punctuation. The kind of interface the QCSAPI is designed for typically follows "qcsapi" - e.g. "interface" for a general (network) interface or "wifi" for an API that only applies for WiFI devices (wifi0, etc.). Next usually is "get" or "set" to show whether the API returns a parameter value or configures the parameter value. The last part of the name describes the parameter the API works with.

5.1.2 QCSAPIs data types and data structs

Many of the QCSAPI data types and data structs follow the definitions in the TR-098 standards. Each datatype is introduced using its literal name or definition as found in the qcsapi.h header file, followed by a description or explanation.
The first API data type provides a platform independent definition of an unsigned 32-bit integer:
`qcsapi_unsigned_int`
This tracks the "unsignedInt" data type from TR-098 and should match the uint32_t type defined in the C99 standard.

The next data type is an enumeration that represents the possible modes of a WiFi device, Access Point or Station.

```
typedef enum {
   qcsapi_mode_not_defined = 1,
   qcsapi_access_point,
   qcsapi_station,
   qcsapi_nosuch_mode = 0
} qcsapi_wifi_mode;
```

Mode not defined is valid; it means the WiFi device has not been configured as an AP or a Station. No such mode is a placeholder for invalid WiFi modes.

The next enumeration represents possible configuration options:

```
typedef enum {
   qcsapi_DFS,
   qcsapi_wmm,
   qcsapi_beacon_advertise,
```

```
   qcsapi_wifi_radio,
   qcsapi_autorate_fallback,
   qcsapi_security,
   qcsapi_SSID_broadcast,
   qcsapi_802_11d,
   qcsapi_wireless_isolation,
   qcsapi_short_GI,
   qcsapi_dfs_fast_channel_switch,
   qcsapi_dfs_no_dfs_scan,
   qcsapi_nosuch_option = 0
} qcsapi_option_type;
```

These are parameters with only two values, yes or true, represented as 1 (actually any non-zero value); and no or false, represented as 0. Only the Get Option and Set Option APIs work with this enum.
The next enumeration represents counters available through the Get Counter API:

```
typedef enum {
        qcsapi_nosuch_counter = 0,
        qcsapi_total_bytes_sent = 1,
        qcsapi_total_bytes_received,
        qcsapi_total_packets_sent,
        qcsapi_total_packets_received,
qcsapi_discard_packets_sent,
        qcsapi_discard_packets_received,
        qcsapi_error_packets_sent,
        qcsapi_error_packets_received,
} qcsapi_counter_type;
```

This enum is one of the required arguments to qcsapi_interface_get_counter.

The next data type represents a 48-bit (6-byte) MAC address:
`qcsapi_mac_addr`

This is NOT a string; a MAC address can have an embedded NUL (value is 0) byte; nor is it required that a NUL byte be present. When setting a MAC address, the 1st 6 bytes will be used to set the MAC address; when getting a MAC address, 6 bytes must be available to accept the address.
These datatypes describe strings of various lengths:

```
typedef char    string_16[ 17 ];
typedef char    string_32[ 33 ];
typedef char    string_64[ 65 ];
typedef char    string_128[ 129 ];
typedef char    string_256[ 257 ];
typedef char    string_1024[ 1025 ];
```

They are provided as a convenience. The reference standards define selected parameters to be strings of fixed length. The internal definition adds one character to insure room for the terminating NUL character; e.g. a string_32 actually has 33 characters.

All string parameters passed to the QCSAPIs are required to be terminated with a NUL character. This includes SSIDs and MCS rates. Any string returned by a QCSAPI will be terminated with a NUL character.

This datatype represents a Service Set identifier:
`qcsapi_SSID`

One additional character is allocated to provide space for a NUL byte to terminate the string. By standard, the SSID can have up to 32 characters. An SSID passed to an API is required to be terminated with a NUL byte.

Next is a datatype to represent the 802.11n paradigm for specifying and setting rates:
`qcsapi_mcs_rate`

It is a string that starts with the letters "MCS", followed by the MCS rate selection. Currently MCS0 to MCS76 (excluding MCS32) are available. Embedded '0's are NOT permitted; "MCS01" is NOT valid.

5.1.3 API signature

"Signature" here refers to a QCSAPI's return value and its arguments.

The return value is always an integer, and always represents the status of the operation. Following the POSIX standard a return value of 0 or positive reports success; a value less that 0 reports an error. If the value is less than 0, it will represent the error. By changing the algebraic sign - by rendering the return value positive, the nature of the error can be determined from the "errno" facility.

An API that returns a parameter value, typically a get API, will return that value in one of the arguments in the argument list. A parameter value will not ever be returned as the value from a QCSAPI.

The first argument is usually the interface, the device the API is to work with. An interface is required to distinguish between an ethernet interface - "eth1_0" and a WiFi interface - "wifi0". And even those APIs targeted for the WiFi interface require the actual interface to distinguish between different Virtual Access Points (VAP).

Several QCSAPIs are generic, in that the API itself works with a class of parameter. Examples include options - parameters with two values, "yes" and "no" or "true" and "false" - and counters - the number of bytes received, or packets transmitted, etc. For these APIs, the second parameter selects the exact parameter - the desired option or counter.

The final argument is usually the value of the parameter the API is working with. For most QCSAPIs, the API itself selects this parameter. For the generic APIs, the second argument selects this parameter. For a SET API, an API that configures a new value, the parameter argument is passed by value; for a GET API, an API that returns the current value of a parameter, the parameter argument is passed by reference.

The following code fragment illustrates a recommended way of calling a QCSAPI and processing the result (notice because this is a GET API, the parameter argument is a reference).

```
qcsapi_result = qcsapi_interface_get_status( "eth1_0", &eth1_status );
if (qcsapi_result < 0) {
        qcsapi_errno = -qcsapi_result;
} else {
        /* call was successful*/
}
```

5.1.4 QCSAPI return values

As stated previously, a return value of 0 or greater than 0 reports success. A return value less than 0 reports an error. The nature of the error is encoded in the return value, and is based on the ERRNO facility from the POSIX standard.

Note

ERRNO values and other API error definitions are positive integers, so programming will need to change the sign of a QCSAPI error return value before comparing with any predefined error definitions.

Please see enum qcsapi_errno for details of all the different error return values that QCSAPI calls may return.

5.1.5 Production mode vs calibration mode

The WiFi device can operate in 2 different modes. Usually the device operates in production mode. In this mode the AP broadcasts beacons and accepts associations from properly qualified STA devices, snd the STA scans the WiFi channels searching for an AP to associate with.

An additional runtime mode, bringup and calibration mode (or calibration mode for short), is available for testing and calibrating the RF stage of the device during the development phase as well as during the manufacturing phase.

The choice between production and calibration mode is made when the device first starts up, based on the value of the boot configuration environmental variable, `calstate`.

If calstate is set to 1, the device operates in calibration mode; otherwise, the device operates in production mode.

Selected APIs that assist with configuring the system are only available in calibration mode. This is noted in the detailed description for each API that has this restriction. Also note that in calibration mode, many of the APIs will not be available.

Please review the writeup on individual APIs before using a particular API in this mode.

In calibration mode, the expected error code for an API that is not available is `-ENODEV`, since those APIs require the name of the WiFi interface, or VAP (Virtual Access Point), which will not be present if the device is running in calibration mode.

5.1.6 Permissions and Access

Selected APIs require root access; that is the user ID of the calling process must be 0.

Chapter 6

Demonstration application: call_qcsapi

An application is present to demonstrate the QCSAPI, `call_qcsapi`. It is a non-interactive command that takes all its parameter from the command line. This application is also the interface to the QSCAPI for scripting applications including web-based GUIs.

6.1 Command Line Parameters

The first command-line parameter specifies the API to be called. Remaining command-line parameters specify the arguments to that API. The order is the order of the arguments in the API. Thus the 2nd argument is usually the interface - e.g. wifi0 or eth1_0. For a generic QSCAPI, the 3rd argument will select the generic parameter - counter, option, etc. A set API will expect the new value as its final parameter.

6.1.1 Format for a MAC address

MAC addresses are 48 bits, or 6 bytes long. They should be entered in this format:
`xx:xx:xx:xx:xx:xx`
where each "xx" represents the value of the corresponding byte in hexadecimal radix. The `call_qcsapi` command uses this same format to display MAC addresses, e.g. Get MAC Address (qcsapi_interface_get_mac_addr) or Get BSSID (qcsapi_wifi_get_BSSID).

Not all sequences of 6 bytes represent valid MAC addresses for an individual interface.

If the low-order bit is set in the 1st byte, the MAC address is interpreted as a multicast address. As this is a logical identifier for a group of interfaces on a network, it is not suitable as the address for an individual interface. Notice the broadcast address, `FF:FF:FF:FF:FF:FF`, is a special case of a multicast address.

The MAC address that is all zeros (`00:00:00:00:00:00`) is also not valid as the MAC address for an individual interface.

By convention, the all-zero MAC address is reported as the BSSID for a STA that is not in association. Since the BSSID for a STA is the MAC address of its partner AP, this means an AP cannot have a WiFi MAC address that is all zeros. This restriction applies to STAs as well as APs.

All APIs that accept a MAC address as a parameter expect that MAC address to be valid for an individual interface. Thus a multicast address will not be accepted as a MAC address, and the selected API will return Invalid Value (`EINVAL`) as the error code. Additional details about command line parameters are listed with each individual QCSAPI.

6.2 Operation and Output

Using its command line parameters, call_qcsapi calls the selected API, reports on the results and then exits. Its output depends on the nature of the API. For APIs that set a parameter, it displays the message complete. For APIs that get a parameter value, it reports the parameter value.

If the API returns an error, the resulting message resembles the example below:

```
QCS API error 22:  Invalid argument
```

The key is the word error, followed by the numeric error code (notice the actual returned error code will be less than 0, here -22). See section QCS API Return Values for more details on error codes.

The `call_qcsapi` application also provides an interface to the QCSAPIs for scripting and web-based applications.

6.3 Examples

The following examples are documented for the reader's reference.

Example 1: Get the operational mode of a device:

```
quantenna # call_qcsapi get_mode wifi0
Access point
quantenna #
```

Example 2: Disable the RF on a device:

```
quantenna # call_qcsapi rfenable wifi0 0
killing cmdloop hostapd with pid: 275
killing hostapd with pid: 278
[93387.180000] wifi0: station 00:26:86:01:1a:be disassociated
[93387.185000] wifi0: station 00:26:86:01:1a:be disassociated
01:56:27.304088   wifi0    Custom driver event:Dot11Msg:Client removed [00:26:86:01:1A:BE] [Deauthenticated
       - 3 - Deauthenticated because sendin[93387.210000] br0: port 2(wifi0) entering disabled state
g STA is leaving (or has left) IBSS or ESS]
01:56:27.304449   wifi0    Custom driver event:STA-TRAFFIC-STAT
mac=00:26:86:01:1a:be
rx_packets=50036
rx_bytes=13669562
tx_packets=17931
tx_bytes=3282979
01:56:27.306999   wifi0    Custom driver event:Dot11Msg:Client disconnected [00:26:86:01:1A:BE] [Client sent
       disassociation - 8 - Disassociated because sending STA is leaving (or has left) BSS]
01:56:27.307305   wifi0    Expired node:00:26:86:01:1A:BE
complete
quantenna #
```

Note in this example there was a single STA associated, which was removed when the RF was disabled on the AP via `call_qcsapi`.

Chapter 7

QCSAPI and the SDK

The QCSAPI is a key part of the SDK. This section describes how to use the SDK and how to access the QCSAPI binary library from within the SDK.

7.1 SDK Contents

The Quantenna SDK is a Linux based distribution (ARC kernel version 2.6.30). Its contents include:

- Sources:
  - Linux Kernel
  - User-space applications
  - C library programs
  - Toolchain (gcc, etc.)
- Binaries
  - Quantenna MuC firmware
  - Selected Linux drivers

7.2 SDK Make Environment

We have tested the SDK build process on the following distributions:

\ul Fedora release 8 (Werewolf) x86 PC running Linux version 2.6.26.8-57.fc8, with native mode gcc compiler version 4.1.2 (versions 4.1 - 4.4 have been tested) and make version GNU Make 3.81. \ul Ubuntu version 9.10 X86 with kernel version 2.6.31-14-generic. \ul Ubuntu version 11.04 X86 SMP with kernel version 2.6.38-8-generic.

The ARC cross compiler for the Quantenna target is included in the SDK. The toolchain is created during the make fromscratch process shown below.

7.3 Installing the SDK

The SDK is delivered as a compressed TAR archive. Download the archive onto the system where the build is to be completed. It is recommended to store the archive in an empty folder, although since the SDK TAR archive stores all files in its own subfolder, an empty folder is not absolutely required.

After unpacking the archive, change directory to the SDK subfolder, e.g.
`quantenna-sdk-021009`
Now enter this command:
`make fromscratch`

This completes the first comprehensive build and will take a significant amount of time, say 30 minutes. At the end of the make process, a new subfolder within the SDK subfolder will be present: `tftp`. This subfolder has the file `ruby-linux.img` (uncompressed image file), and `ruby-linux.img.lzma` (compressed image file).

7.4 QCSAPI components in the SDK

The QCSAPI has two components, the header file and the binary library. In addition the application / interface program, call_qcsapi is included. Each is built or installed as part of the overall SDK build process.

The library file will be installed in `buildroot/build_arc/staging_dir/lib`, (path relative to the top-level SDK folder, e.g. quantenna-sdk-021009) as this is the folder the cross compiler searches by default for binary libraries. Thus to build an application that calls the QCSAPI, it is only necessary to pass the option `-lqcsapi` to the cross-linker.

The header file will be installed in buildroot/include. Thus to locate this include file, when building an application to call QCSAPIs, include the path to this folder.

The application `call_qcsapi` will be present in `/sbin/` when the Quantenna device boots up with a build from the SDK. Within the SDK, the contents of the QCSAPI are available in the subdirectory `buildroot/package/qcsapi/qcsapi-1.0.1/`

Chapter 8

Using QCSAPI via RPC

8.1 Introduction

8.1.1 rpcgen

`rpcgen` is a tool for creating remote procedure call stubs in C, given an RPC protocol definition file. These stubs contain code relevant for marshaling and un-marshaling, which is serialization of data so that it can be transmitted over a network. The marshaled data can be sent over UDP sockets, TCP sockets, or through PCIe.

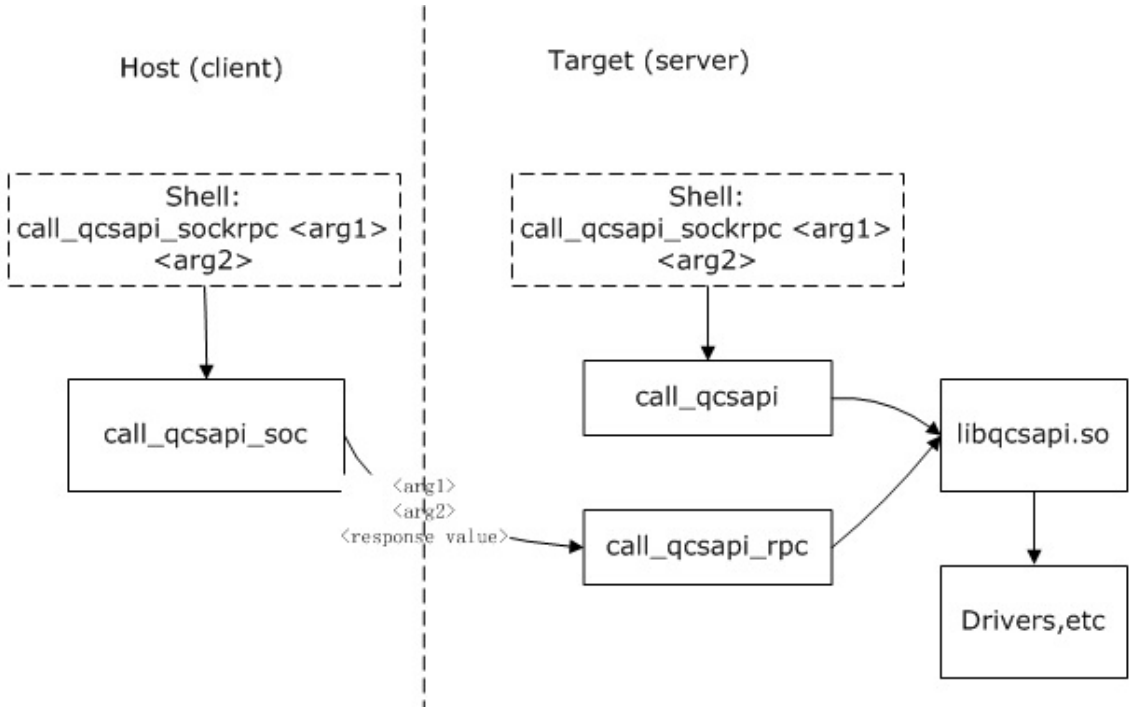
8.2 call_qcsapi_sockrpc / call_qcsapi_rpcd

These are utilities for remote invocation of the `call_qcsapi` example program.

`call_qcsapi_rpcd` is a server implementation, which is much like `call_qcsapi`, except that it runs as a daemon, accepting requests from RPC clients. Requests take the format of a list of strings, which are equivalent to `argv` in `call_qcsapi`.

`call_qcsapi_sockrpc` is a client implementation, which finds the server hostname, then creates a request out of argc and `argv`, waits for a response then prints it. It uses UDP sockets to communicate with the server.

`call_qcsapi_pcie` is a client implementation which creates requests from `argc` and `argv`, but it uses PCIe to communicate with the server instead of UDP sockets.



8.2.1 Implementation

`call_qcsapi_rpc` and associated client programs use `rpcgen` to create client and server stubs based on a simple interface definition file, `call_qcsapi_rpc.x`.

8.3 RPC for qcsapi.h

This is an RPC service, which provides and RPC interface for (almost) all of the functions in libqcsapi.

8.3.1 libqcsapi

The target device includes a binary library, `libqcsapi.so`, which provides implementations for the functions prototyped in `qcsapi.h`. Third party developers may develop applications that run on the target and link to libqcsapi.so

8.3.2 libqcsapi_client

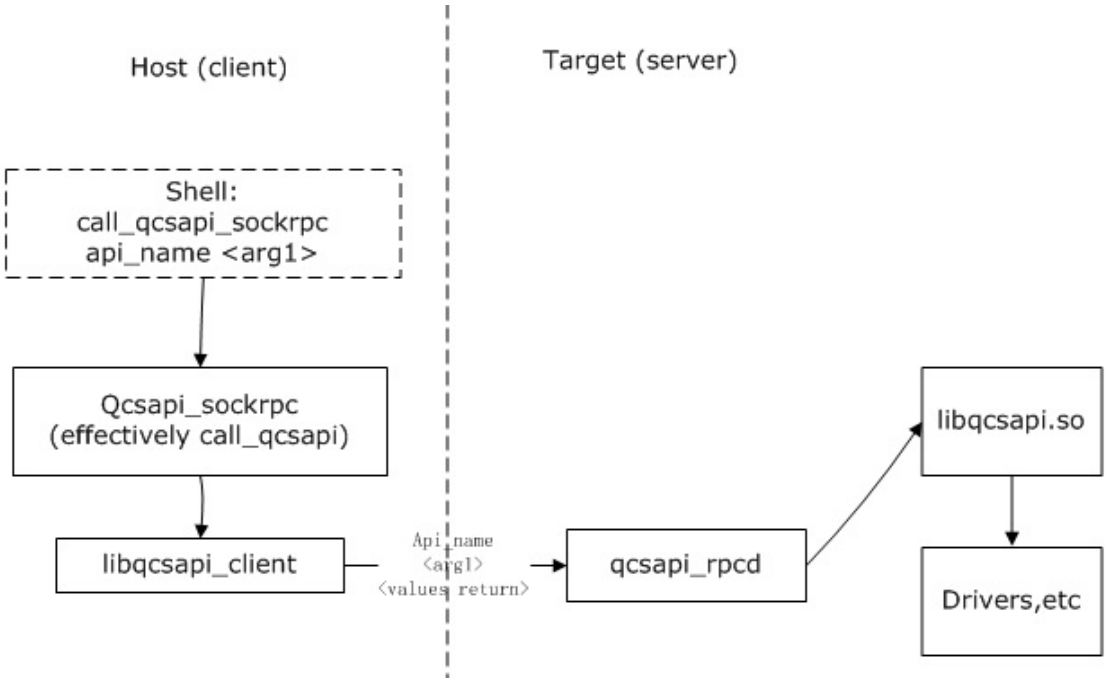
In order to develop applications that can use QCSAPI, but run on another host, libqcsapi_client provides RPC stubs based on prototypes in qcsapi.h, with an additional function for specifying the RPC transport mechanism (UDP/T↩
CP/PCIe).
Third party developers can create a program running on a host platform, link against `libqcsapi_client.so`, and call qcsapi functions from C code.
`libqcsapi_client.so` communicates with server RPC stubs running on `qcsapi_rpcd`.

8.3.3 qcsapi_rpcd

qcsapi_rpcd is an RPC service program, which contains RPC server stubs for each function in libqcsapi. qcsapi_rpcd registers UDP, TCP and PCIe services. It links to libqcsapi.so, to invoke the real QCS↩ API functions when requested by clients.

8.3.4 qcsapi_sockrpc

qcsapi_sockrpc is a version of call_qcsapi, which is linked against libqcsapi_client.so instead of libqcsapi.so. It works as follows:



8.4 Implementation details of libqcsapi_client + qcsapi_rpcd

To reduce ongoing maintenance effort required, libqcsapi_client and qcsapi_rpcd are both automatically generated based on the contents of qcsapi.h; it is just RPC client and server stubs. The process of code generation is as follows. All of these files are relative paths under buildroot/package/qcsapi/qcsapi-1.↩ 0.1/.

1. qcsapi.h is manually changed with new feature developments, as it always has been.
2. qcsapi_rpc/qcsapi_rpc_gen.pl reads qcsapi.h, and generates:
   (a) an rpc interface definition file for use with rpcgen qcsapi_rpc/generated/qcsapi_rpc.x. This file is later used as an input for rpcgen.
   (b) client stub implementations: qcsapi_rpc/generated/qcsapi_rpc_clnt_adapter.c. This will contain function definitions matched to the prototypes in qcsapi.h, which convert qcsapi.h calls to the relevant RPC routines.

(c) server stub adapter implementation:
qcsapi_rpc/generated/qcsapi_rpc_svc_adapter.c, which convert from rpcgen RPC server stub functions to qcsapi.h calls; these calls will go to the real implementation in libqcsapi.so.

3. Once the perl script generates qcsapi_rpc/generated/qcsapi_rpc.x, rpcgen is used to create other files:

   (a) qcsapi_rpc/generated/qcsapi_rpc.h, structure definitions and function prototypes to represent data that is appropriate for marshaling/un-marshaling.

   (b) qcsapi_rpc/generated/qcsapi_rpc_clnt.c, which is rpcgen generated client stubs, but these are unused. Required code is already in qcsapi_rpc_clnt_adapter.c.

   (c) qcsapi_rpc/generated/qcsapi_rpc_svc.c, RPC service function demultiplexer. Takes arguments from the RPC service program, and will call the appropriate function in qcsapi_rpc_svc_adapter.c.

   (d) qcsapi_rpc/generated/qcsapi_rpc_xdr.c marshaling/un-marshaling functions.

4. Programs are compiled and linked with the following dependencies:

   (a) qcsapi_rpcd
      i. qcsapi_rpc/generated/qcsapi_rpc_svc.c
      ii. qcsapi_rpc/generated/qcsapi_rpc_svc_adapter.c
      iii. qcsapi_rpc/generated/qcsapi_rpc_xdr.c
      iv. libqcsapi.so
      v. Additional code for starting and registering the server, and PCIe server transport.

   (b) libqcsapi_client
      i. qcsapi_rpc/generated/qcsapi_rpc_clnt_adapter.c
      ii. qcsapi_rpc/generated/qcsapi_rpc_xdr.c

   (c) qcsapi_sockrpc:
      i. Code for resolving which remote server to use
      ii. call_qcsapi frontend code
      iii. libqcsapi_client.so

8.5 QCSAPI RPC in the SDK

The sdk ships with generated sources for qcsapi_rpc and call_qcsapi_rpc, but not with qcsapi_rpc/qcsapi_rpc_gen.pl, or interface definition files. This allows libqcsapi_client and other client programs to be rebuild by customers, but any changes to qcsapi.h will not automatically update the RPC code.

8.5.1 Source bundles

call_qcsapi clients and qcsapi_rpc clients (including libqcsapi_client.so source) are included in the SDK as source code .zip bundles, which can be taken integrated with host tools by vendors, so their host firmware can contain QCSAPI RPC client programs without the Quantenna build system. A caveat of this is that automatically generated code based on qcsapi.h will be updated with each change to qcsapi.h, so vendors must be diligent to make sure they are matching the appropriate client release with the Quantenna target platform server.

Chapter 9

QCSAPI detailed documentation

9.1 Overview and conventions

This chapter gives an overview of how the data structures and APIs are documented. An example function prototype is shown directly below.

- int call_qcsapi_example (const char *ifname, int example_input, int *example_output)

  A brief description of the function call is provided inline with the function listing for each section.

9.1.1 Detailed Description

This chapter details the data structures (structs, enums, types etc.) used by the QCSAPI, as well as the detailed information on the APIs.

APIs are documented fully with details of what the functional call does, the parameters accepted (input/outputs) and the return values.

Each API also has a brief detail of if and how the function can be called using the `call_qcsapi` command line utility. Some APIs are not able to be called using `call_qcsapi` through the nature of the API.

This chapter is divided into the data structure detailed documentation, followed by subsections detailing rough functional API areas.

The following section gives an example of how an API call is documented.

9.1.2 Function Documentation

9.1.2.1 call_qcsapi_example()

```
int call_qcsapi_example (
            const char * ifname,
            int example_input,
            int * example_output )
```

In the Function Documentation section, for each API call, there is a detailed definition of the function call, with extra information, side effects, pre-requisites etc. This appears in the section which documents the individual API call.

After the detailed documentation the list of input/output parameters is given.

Parameters

ifname	Details of the parameter 'ifname'
example_input	Details of the input parameter 'example_input'
example_output	Details of the output parameter 'example_output'. Output parameters are generally seen as pointers to variables.

After the parameter list, the return values are documented.

Returns

Details of the return value (generally >= 0 indicates success, < 0 indicates failure). See the section QCSAPI return values for details of the different return values.

In addition to this, there may be extra documentation and references to other function calls.

Note

Something noteworthy about the API may be documented in one of these blocks.

Warning

Something that the reader **must** read and take into account may be documented in one of these blocks.

`call_qcsapi` interface:

This is where the command line `call_qcsapi` interface is detailed. Input parameters, and expected output will be given.

Note

Not all QCSAPI C API calls have equivalent `call_qcsapi` command line calls.

9.2 Detailed data type and enumeration documentation

This section contains detailed documentation on the data types and enumerations used in the QCSAPI.

Data Structures

- struct qcsapi_data_64bytes

  Convenience definition to represent a 64 byte array.
- struct qcsapi_data_128bytes

  Convenience definition to represent a 128 unsigned byte array.
- struct qcsapi_data_256bytes

  Convenience definition to represent a 256 unsigned byte array.
- struct qcsapi_data_512bytes

  Convenience definition to represent a 512 unsigned byte array.
- struct qcsapi_data_1Kbytes

  Convenience definition to represent a 1024 unsigned byte array.
- struct qcsapi_data_2Kbytes

  Convenience definition to represent a 2048 unsigned byte array.
- struct qcsapi_data_3Kbytes

  Convenience definition to represent a 3072 unsigned byte array.
- struct qcsapi_data_4Kbytes

  Convenience definition to represent a 4096 unsigned byte array.
- struct qcsapi_int_array32

  Convenience definition to represent a 32 integer array.
- struct qcsapi_int_array128

  Convenience definition to represent a 128 integer array.
- struct qcsapi_int_array256

  Convenience definition to represent a 256 integer array.
- struct qcsapi_int_array768

  Convenience definition to represent a 768 integer array.
- struct qcsapi_int_array1024

  Convenience definition to represent a 1024 integer array.
- struct qcsapi_log_param

  Struct to store parameters for respective module.
- struct qcsapi_channel_power_table

  Structure to contain the power table for a single channel.
- struct qcsapi_ap_properties

  This structure represents a set of properties for a single AP.
- struct qcsapi_node_stats

  Structure to contain per node statistics.
- struct _qcsapi_interface_stats

  Structure to contain per interface statistics.
- struct _qcsapi_phy_stats

  Structure containing PHY statistics.
- struct _qcsapi_mlme_stats

  Structure containing per client mlme statistics.
- struct _qcsapi_mlme_stats_macs

  Structure containing the list of macs.
- struct qcsapi_cca_info
- struct qcsapi_scs_currchan_rpt

Structure containing SCS report for current channel.

- struct qcsapi_scs_ranking_rpt

  Structure containing SCS report for all channels.

- struct qcsapi_scs_score_rpt

  Structure containing the scores of all channels.

- struct qcsapi_autochan_rpt

  Structure containing auto channel report for initial channel selection.

- struct qcsapi_scs_param_rpt
- struct qcsapi_assoc_records
- struct _qcsapi_node_txrx_airtime
- struct _qcsapi_csw_record
- struct _qcsapi_dscp2ac_data
- struct _qcsapi_chan_disabled_data
- struct _qcsapi_radar_status
- struct _qcsapi_disconn_info
- struct _qcsapi_calcmd_tx_power_rsp
- struct _qcsapi_calcmd_rssi_rsp
- struct qcsapi_measure_basic_s
- struct qcsapi_measure_cca_s
- struct qcsapi_measure_rpi_s
- struct qcsapi_measure_chan_load_s
- struct qcsapi_measure_noise_his_s
- struct qcsapi_measure_beacon_s
- struct qcsapi_measure_frame_s
- struct qcsapi_measure_tran_steam_cat_s
- struct qcsapi_measure_multicast_diag_s
- union _qcsapi_measure_request_param
- struct qcsapi_measure_neighbor_item_s
- struct qcsapi_measure_rpt_tpc_s
- struct qcsapi_measure_rpt_noise_histogram_s
- struct qcsapi_measure_rpt_beacon_s
- struct qcsapi_measure_rpt_frame_s
- struct qcsapi_measure_rpt_tran_stream_cat_s
- struct qcsapi_measure_rpt_multicast_diag_s
- struct qcsapi_measure_rpt_tpc_report_s
- struct qcsapi_measure_rpt_link_measure_s
- struct qcsapi_measure_rpt_neighbor_report_s
- union _qcsapi_measure_report_result
- struct qcsapi_mac_list
- struct qcsapi_sample_assoc_data
- struct _qcsapi_vlan_config

Macros

- #define **QCSAPI_LOCAL_NODE** 0
- #define **QCSAPI_REMOTE_NODE** 1
- #define **QCSAPI_TRUE** 1
- #define **QCSAPI_FALSE** 0
- #define MAX_NUM_OF_BANDWIDTHS 5
- #define **QCSAPI_POWER_TOTAL** 8 /∗ the total power indices in a channel power table ∗/
- #define **MACFILTERINGMACFMT** "%02x:%02x:%02x:%02x:%02x:%02x"
- #define **MAC_ADDR_SIZE** ETHER_ADDR_LEN
- #define **MAC_ADDR_STRING_LENGTH** 18

- #define **MAC_ADDR_LIST_SIZE** 8
- #define **QCSAPI_SSID_MAXLEN** (IW_ESSID_MAX_SIZE + 1)
- #define **QCSAPI_SSID_MAXNUM** 32
- #define **QCSAPI_STATUS_MAXLEN** 12
- #define **QCSAPI_SSID_MAX_RECORDS** (6)
- #define **QCSAPI_MCS_RATE_MAXLEN** 8
- #define **IEEE80211_PROTO_11B** 0x00000001
- #define **IEEE80211_PROTO_11G** 0x00000002
- #define **IEEE80211_PROTO_11A** 0x00000004
- #define **IEEE80211_PROTO_11N** 0x00000008
- #define **IEEE80211_PROTO_11AC** 0x00000010
- #define **IEEE80211_WMM_AC_BE** 0 /∗ best effort ∗/
- #define **IEEE80211_WMM_AC_BK** 1 /∗ background ∗/
- #define **IEEE80211_WMM_AC_VI** 2 /∗ video ∗/
- #define **IEEE80211_WMM_AC_VO** 3 /∗ voice ∗/
- #define **IEEE8021P_PRIORITY_ID0** 0
- #define **IEEE8021P_PRIORITY_ID1** 1
- #define **IEEE8021P_PRIORITY_ID2** 2
- #define **IEEE8021P_PRIORITY_ID3** 3
- #define **IEEE8021P_PRIORITY_ID4** 4
- #define **IEEE8021P_PRIORITY_ID5** 5
- #define **IEEE8021P_PRIORITY_ID6** 6
- #define **IEEE8021P_PRIORITY_ID7** 7
- #define **IEEE8021P_PRIORITY_NUM** 8
- #define **IEEE80211_QOS_MAP_DSCP_EXCEPT_MAX** 21
- #define **IEEE80211_QOS_MAP_MAX** 255
- #define **QCSAPI_WIFI_AC_MAP_SIZE** (64)
- #define **IP_DSCP_NUM** 64
- #define **QCSAPI_MLME_STATS_MAX_MACS** 128
- #define **QCSAPI_SCS_REPORT_CHAN_NUM** 32
- #define **QCSAPI_ASSOC_MAX_RECORDS** 32
- #define **TABLE_SIZE**(TABLE) (sizeof(TABLE) / sizeof((TABLE)[0]))
- #define **QCSAPI_ANY_CHANNEL** 0
- #define **QCSAPI_MIN_CHANNEL** 1
- #define **QCSAPI_MAX_CHANNEL** IEEE80211_CHAN_MAX
- #define **RESTORE_DEFAULT_CONFIG** "/scripts/restore_default_config"
- #define **BRIDGE_DEVICE** "br0"
- #define **QCSAPI_CSW_MAX_RECORDS** 32
- #define QCSAPI_MAX_MACS_IN_LIST 200
- #define **QCSAPI_MAX_MACS_SIZE** 1200
- #define **QCSAPI_NUM_ANT** 5

Typedefs

- typedef int qcsapi_int_a32[32]
- typedef uint32_t qcsapi_unsigned_int
- typedef uint64_t qcsapi_unsigned_int64
- typedef uint8_t qcsapi_mac_addr[MAC_ADDR_SIZE]

  Convenience definition to represent a 6 byte MAC address.
- typedef uint8_t **qcsapi_mac_addr_list**[MAC_ADDR_LIST_SIZE ∗MAC_ADDR_SIZE]
- typedef char qcsapi_SSID[QCSAPI_SSID_MAXLEN]

  Convenience definition for a string large enough for a single SSID.
- typedef char qcsapi_mcs_rate[QCSAPI_MCS_RATE_MAXLEN]

Type used to contain an MCS definition.

- typedef char string_16[17]

  Convenience definition for a string of size 16.
- typedef char string_32[33]

  Convenience definition for a string of size 32.
- typedef char string_64[65]

  Convenience definition for a string of size 64.
- typedef char string_128[129]

  Convenience definition for a string of size 128.
- typedef char string_256[257]

  Convenience definition for a string of size 256.
- typedef char string_512[513]

  Convenience definition for a string of size 512.
- typedef char string_1024[1025]

  Convenience definition for a string of size 1024.
- typedef char string_2048[2049]

  Convenience definition for a string of size 2048.
- typedef char string_4096[4097]

  Convenience definition for a string of size 4096.
- typedef struct qcsapi_channel_power_table **qcsapi_channel_power_table**
- typedef struct qcsapi_ap_properties qcsapi_ap_properties

  This structure represents a set of properties for a single AP.
- typedef struct qcsapi_node_stats qcsapi_node_stats

  Structure to contain per node statistics.
- typedef struct _qcsapi_interface_stats qcsapi_interface_stats

  Structure to contain per interface statistics.
- typedef struct _qcsapi_phy_stats qcsapi_phy_stats

  Structure containing PHY statistics.
- typedef struct _qcsapi_mlme_stats qcsapi_mlme_stats

  Structure containing per client mlme statistics.
- typedef struct _qcsapi_mlme_stats_macs qcsapi_mlme_stats_macs

  Structure containing the list of macs.
- typedef struct qcsapi_scs_currchan_rpt qcsapi_scs_currchan_rpt

  Structure containing SCS report for current channel.
- typedef struct qcsapi_scs_ranking_rpt qcsapi_scs_ranking_rpt

  Structure containing SCS report for all channels.
- typedef struct qcsapi_scs_score_rpt qcsapi_scs_score_rpt

  Structure containing the scores of all channels.
- typedef struct qcsapi_autochan_rpt qcsapi_autochan_rpt

  Structure containing auto channel report for initial channel selection.
- typedef struct qcsapi_scs_param_rpt qcsapi_scs_param_rpt
- typedef struct qcsapi_assoc_records qcsapi_assoc_records
- typedef struct _qcsapi_node_txrx_airtime qcsapi_node_txrx_airtime
- typedef struct _qcsapi_csw_record qcsapi_csw_record
- typedef struct _qcsapi_dscp2ac_data qcsapi_dscp2ac_data
- typedef struct _qcsapi_chan_disabled_data **qcsapi_chan_disabled_data**
- typedef struct _qcsapi_radar_status qcsapi_radar_status
- typedef struct _qcsapi_disconn_info qcsapi_disconn_info
- typedef struct _qcsapi_calcmd_tx_power_rsp qcsapi_calcmd_tx_power_rsp
- typedef struct _qcsapi_calcmd_rssi_rsp qcsapi_calcmd_rssi_rsp
- typedef union _qcsapi_measure_request_param qcsapi_measure_request_param
- typedef union _qcsapi_measure_report_result qcsapi_measure_report_result
- typedef struct _qcsapi_vlan_config qcsapi_vlan_config

Enumerations

- enum qcsapi_errno {
  qcsapi_errno_base = 1000,
  qcsapi_system_not_started = qcsapi_errno_base,
  qcsapi_parameter_not_found = qcsapi_errno_base + 1,
  qcsapi_SSID_not_found = qcsapi_errno_base + 2,
  qcsapi_only_on_AP = qcsapi_errno_base + 3,
  qcsapi_only_on_STA = qcsapi_errno_base + 4,
  qcsapi_configuration_error = qcsapi_errno_base + 5,
  qcsapi_buffer_overflow = qcsapi_errno_base + 6,
  qcsapi_internal_format_error = qcsapi_errno_base + 7,
  qcsapi_programming_error = qcsapi_errno_base + 8,
  qcsapi_bringup_mode_only = qcsapi_errno_base + 9,
  qcsapi_daemon_socket_error = qcsapi_errno_base + 10,
  qcsapi_conflicting_options = qcsapi_errno_base + 11,
  qcsapi_SSID_parameter_not_found = qcsapi_errno_base + 12,
  qcsapi_not_initialized = qcsapi_errno_base + 13,
  qcsapi_invalid_type_image_file = qcsapi_errno_base + 14,
  qcsapi_image_file_failed_chkimage = qcsapi_errno_base + 15,
  qcsapi_flash_partition_not_found = qcsapi_errno_base + 16,
  qcsapi_erase_flash_failed = qcsapi_errno_base + 17,
  qcsapi_copy_image_flash_failed = qcsapi_errno_base + 18,
  qcsapi_invalid_wifi_mode = qcsapi_errno_base + 19,
  qcsapi_process_table_full = qcsapi_errno_base + 20,
  qcsapi_measurement_not_available = qcsapi_errno_base + 21,
  qcsapi_too_many_bssids = qcsapi_errno_base + 22,
  qcsapi_only_on_primary_interface = qcsapi_errno_base + 23,
  qcsapi_too_many_wds_links = qcsapi_errno_base + 24,
  qcsapi_config_update_failed = qcsapi_errno_base + 25,
  qcsapi_no_network_counters = qcsapi_errno_base + 26,
  qcsapi_invalid_pm_interval = qcsapi_errno_base + 27,
  qcsapi_only_on_wds = qcsapi_errno_base + 28,
  qcsapi_only_unicast_mac = qcsapi_errno_base + 29,
  qcsapi_primary_iface_forbidden = qcsapi_errno_base + 30,
  qcsapi_invalid_ifname = qcsapi_errno_base + 31,
  qcsapi_iface_error = qcsapi_errno_base + 32,
  qcsapi_sem_error = qcsapi_errno_base + 33,
  qcsapi_not_supported = qcsapi_errno_base + 34,
  qcsapi_invalid_dfs_channel = qcsapi_errno_base + 35,
  qcsapi_script_error = qcsapi_errno_base + 36,
  qcsapi_invalid_wds_peer_addr = qcsapi_errno_base + 37,
  qcsapi_band_not_supported = qcsapi_errno_base + 38,
  qcsapi_region_not_supported = qcsapi_errno_base + 39,
  qcsapi_region_database_not_found = qcsapi_errno_base + 40,
  qcsapi_param_name_not_supported = qcsapi_errno_base + 41,
  qcsapi_param_value_invalid = qcsapi_errno_base + 42,
  qcsapi_invalid_mac_addr = qcsapi_errno_base + 43,
  qcsapi_option_not_supported = qcsapi_errno_base + 44,
  qcsapi_wps_overlap_detected = qcsapi_errno_base + 45,
  qcsapi_mlme_stats_not_supported = qcsapi_errno_base + 46,
  qcsapi_board_parameter_not_supported = qcsapi_errno_base + 47,
  qcsapi_peer_in_assoc_table = qcsapi_errno_base + 48,
  qcsapi_mac_not_in_assoc_list = qcsapi_errno_base + 49,
  qcsapi_param_count_exceeded = qcsapi_errno_base + 50,
  qcsapi_duplicate_param = qcsapi_errno_base + 51,
  qcsapi_iface_invalid = qcsapi_errno_base + 52,
  qcsapi_exceed_config_number = qcsapi_errno_base + 53 }

  This enumeration represents the internal QCSAPI error return values that may be returned by various QCSAPIs.

- enum qcsapi_led {
  qcsapi_AGPIO1_LED = 1,
  qcsapi_AGPIO2_LED = 2,
  qcsapi_AGPIO3_LED = 3,
  qcsapi_AGPIO4_LED = 4,
  qcsapi_AGPIO5_LED = 5,
  qcsapi_AGPIO7_LED = 7,
  qcsapi_AGPIO11_LED = 11,
  qcsapi_AGPIO12_LED = 12,
  qcsapi_AGPIO27_LED = 27,
  qcsapi_nosuch_GPIO = 255,
  QCSAPI_MAX_LED = 31 }

  This enumeration represents an abstract LED value.
- enum qcsapi_auth_crypto {
  qcsapi_protocol_WPA_mask = 1,
  qcsapi_protocol_11i_mask = 2,
  qcsapi_ap_PSK_authentication = 1,
  qcsapi_ap_EAP_authentication = 2,
  qcsapi_ap_TKIP_encryption_mask = 0x01,
  qcsapi_ap_CCMP_encryption_mask = 0x02,
  qcsapi_ap_security_enabled = 0x01 }

  This enumeration represents a set of security and authentication modes.
- enum qcsapi_gpio_config {
  qcsapi_gpio_not_available = 0,
  qcsapi_gpio_input_only,
  qcsapi_gpio_output,
  qcsapi_nosuch_gpio_config = -1 }

  This enumeration is used to represent GPIO state.
- enum qcsapi_file_path_config {
  qcsapi_security_configuration_path = 0,
  qcsapi_nosuch_file_path = -1 }

  This enumeration is used to abstract configuration file paths.
- enum qcsapi_wifi_mode {
  qcsapi_mode_not_defined = 1,
  qcsapi_access_point,
  qcsapi_station,
  qcsapi_wds,
  qcsapi_repeater,
  qcsapi_nosuch_mode = 0 }

  This enumeration represents the operational mode of the device.
- enum qcsapi_rate_type {
  qcsapi_basic_rates = 1,
  qcsapi_operational_rates,
  qcsapi_possible_rates,
  qcsapi_nosuch_rate = 0 }

  Enumeration to represent rate sets.
- enum qcsapi_mimo_type {
  qcsapi_mimo_ht = 1,
  qcsapi_mimo_vht,
  qcsapi_nosuch_standard = 0 }

  Enumeration to represent 802.11 standards.
- enum qcsapi_counter_type {
  qcsapi_nosuch_counter = 0,
  QCSAPI_NOSUCH_COUNTER = qcsapi_nosuch_counter,
  qcsapi_total_bytes_sent = 1,
  QCSAPI_TOTAL_BYTES_SENT = qcsapi_total_bytes_sent,

qcsapi_total_bytes_received,
QCSAPI_TOTAL_BYTES_RECEIVED = qcsapi_total_bytes_received,
qcsapi_total_packets_sent,
QCSAPI_TOTAL_PACKETS_SENT = qcsapi_total_packets_sent,
qcsapi_total_packets_received,
QCSAPI_TOTAL_PACKETS_RECEIVED = qcsapi_total_packets_received,
qcsapi_discard_packets_sent,
QCSAPI_DISCARD_PACKETS_SENT = qcsapi_discard_packets_sent,
qcsapi_discard_packets_received,
QCSAPI_DISCARD_PACKETS_RECEIVED = qcsapi_discard_packets_received,
qcsapi_error_packets_sent,
QCSAPI_ERROR_PACKETS_SENT = qcsapi_error_packets_sent,
qcsapi_error_packets_received,
QCSAPI_ERROR_PACKETS_RECEIVED = qcsapi_error_packets_received,
qcsapi_fragment_frames_received,
QCSAPI_FRAGMENT_FRAMES_RECEIVED = qcsapi_fragment_frames_received,
qcsapi_vlan_frames_received,
QCSAPI_VLAN_FRAMES_RECEIVED = qcsapi_vlan_frames_received }

Enumeration used to represent different interface counters.

- enum qcsapi_per_assoc_param {
QCSAPI_NO_SUCH_PER_ASSOC_PARAM = 0,
QCSAPI_LINK_QUALITY = 1,
QCSAPI_RSSI_DBM,
QCSAPI_BANDWIDTH,
QCSAPI_SNR,
QCSAPI_TX_PHY_RATE,
QCSAPI_RX_PHY_RATE,
QCSAPI_STAD_CCA,
QCSAPI_HW_NOISE,
QCSAPI_STA_IP,
QCSAPI_RSSI,
QCSAPI_PHY_NOISE,
QCSAPI_SOC_MAC_ADDR,
QCSAPI_SOC_IP_ADDR,
QCSAPI_NODE_MEAS_BASIC,
QCSAPI_NODE_MEAS_CCA,
QCSAPI_NODE_MEAS_RPI,
QCSAPI_NODE_MEAS_CHAN_LOAD,
QCSAPI_NODE_MEAS_NOISE_HIS,
QCSAPI_NODE_MEAS_BEACON,
QCSAPI_NODE_MEAS_FRAME,
QCSAPI_NODE_MEAS_TRAN_STREAM_CAT,
QCSAPI_NODE_MEAS_MULTICAST_DIAG,
QCSAPI_NODE_TPC_REP,
QCSAPI_NODE_LINK_MEASURE,
QCSAPI_NODE_NEIGHBOR_REP }

Enumeration for parameters as read in via qcsapi_wifi_get_node_param.

- enum qcsapi_option_type {
qcsapi_channel_refresh = 1,
qcsapi_DFS,
qcsapi_wmm,
qcsapi_mac_address_control,
qcsapi_beacon_advertise,
qcsapi_wifi_radio,
qcsapi_autorate_fallback,
qcsapi_security,
qcsapi_SSID_broadcast,
qcsapi_802_11d,

qcsapi_wireless_isolation,
qcsapi_short_GI,
qcsapi_802_11h,
qcsapi_dfs_fast_channel_switch,
qcsapi_dfs_avoid_dfs_scan,
qcsapi_uapsd,
qcsapi_tpc_query,
qcsapi_sta_dfs,
qcsapi_specific_scan,
qcsapi_GI_probing,
qcsapi_GI_fixed,
qcsapi_stbc,
qcsapi_beamforming,
qcsapi_short_slot,
qcsapi_short_preamble,
qcsapi_rts_cts,
qcsapi_40M_only,
qcsapi_obss_coexist,
qcsapi_11g_protection,
qcsapi_11n_protection,
qcsapi_qlink,
qcsapi_sta_dfs_strict,
qcsapi_nosuch_option = 0 }

Enumeration used in the option set/get API.

- enum qcsapi_board_parameter_type {
qcsapi_hw_revision = 1,
qcsapi_hw_id,
qcsapi_hw_desc,
qcsapi_rf_chipid,
qcsapi_bond_opt,
qcsapi_vht,
qcsapi_bandwidth,
qcsapi_spatial_stream,
qcsapi_interface_types,
qcsapi_rf_chip_verid,
qcsapi_nosuch_parameter = 0 }

Enumeration used in the board parameter get API.

- enum qcsapi_service_name {
QCSAPI_SERVICE_MAUI = 0,
QCSAPI_SERVICE_TELNET = 1,
QCSAPI_SERVICE_DHCP_CLIENT = 2,
QCSAPI_SERVICE_HTTPD = 3,
QCSAPI_SERVICE_MONITOR_TEMPERATURE = 4,
QCSAPI_SERVICE_QEVT = 5,
QCSAPI_NOSUCH_SERVICE = -1 }

Enumeration used to find the service index.

- enum qcsapi_service_start_index {
qcsapi_service_maui_start_index = 90,
qcsapi_service_inetd_start_index = 42,
qcsapi_service_dhclient_start_index = 91,
qcsapi_service_httpd_start_index = 92,
qcsapi_service_monitor_temp_start_index = 70,
qcsapi_service_qevt_start_index = 41,
qcsapi_service_no_such_index = -1 }

Enumeration used to map start_index in /etc/init.d/.

- enum qcsapi_service_action {
QCSAPI_SERVICE_START = 0,

QCSAPI_SERVICE_STOP = 1,
QCSAPI_SERVICE_ENABLE = 2,
QCSAPI_SERVICE_DISABLE = 3,
QCSAPI_SERVICE_STATUS = 4,
QCSAPI_NOSUCH_ACTION = -1 }

Enumeration used to find the service action.

- enum qcsapi_log_module_name {
QCSAPI_WPA_SUPPLICANT = 0,
QCSAPI_HOSTAPD = 1,
QCSAPI_KERNEL = 2,
QCSAPI_DRIVER = 3,
QCSAPI_NOSUCH_MODULE = -1 }

  Enumeration used to identify the module to perform log actions on.

- enum qcsapi_remote_log_action {
QCSAPI_REMOTE_LOG_ENABLE = 0,
QCSAPI_REMOTE_LOG_DISABLE = 1,
QCSAPI_NO_SUCH_ACTION = -1 }

  Enumeration used to identify the action for remote logging.

- enum qcsapi_console_action {
QCSAPI_CONSOLE_ENABLE = 0,
QCSAPI_CONSOLE_DISABLE = 1,
QCSAPI_NO_ACTION = -1 }

  Enumeration used to identify the action for console.

- enum qcsapi_bw {
qcsapi_bw_20MHz = 20,
qcsapi_bw_40MHz = 40,
qcsapi_bw_80MHz = 80,
qcsapi_bw_160MHz = 160,
qcsapi_nosuch_bw }

  This enumeration represents the bandwidth in use on the device.

- enum qcsapi_power_indices {
QCSAPI_POWER_INDEX_BFOFF_1SS = 0,
QCSAPI_POWER_INDEX_BFOFF_2SS,
QCSAPI_POWER_INDEX_BFOFF_3SS,
QCSAPI_POWER_INDEX_BFOFF_4SS,
QCSAPI_POWER_INDEX_BFON_1SS,
QCSAPI_POWER_INDEX_BFON_2SS,
QCSAPI_POWER_INDEX_BFON_3SS,
QCSAPI_POWER_INDEX_BFON_4SS }

  This enumeration represents the indices for different spatial streams (1-4) and BF cases (on/off).

- enum qcsapi_pmf {
qcsapi_pmf_disabled = 0,
qcsapi_pmf_optional = 1,
qcsapi_pmf_required = 2 }

  This enumeration represents the 802.11w / PMF capability of the VAP.

- enum qcsapi_11nac_stat {
qcsapi_11nac_disable = 0,
qcsapi_11nac_enable = 1 }

  This enumeration represents the mode in use on the device.

- enum qcsapi_pref_band {
qcsapi_band_2_4ghz = 0,
qcsapi_band_5ghz = 1,
qcsapi_nosuch_band = 2 }

  This enumeration represents the preferred band to use on the device.

- enum qcsapi_rf_chip_id {
  qcsapi_rf_chipid_2_4ghz = 0,
  qcsapi_rf_chipid_5ghz = 1,
  qcsapi_rf_chipid_dual = 2,
  qcsapi_nosuch_chipid = 3 }

  This enumeration represents the current RF Chip ID.
- enum qcsapi_phy_mode {
  qcsapi_phy_mode_11b = 0,
  qcsapi_phy_mode_11g = 1,
  qcsapi_phy_mode_11ng = 2,
  qcsapi_phy_mode_11ng40 = 3,
  qcsapi_phy_mode_11a = 4,
  qcsapi_phy_mode_11na = 5,
  qcsapi_phy_mode_11na40 = 6,
  qcsapi_phy_mode_11ac20 = 7,
  qcsapi_phy_mode_11ac40 = 8,
  qcsapi_phy_mode_11ac80Edgeplus = 9,
  qcsapi_phy_mode_11ac80Cntrplus = 10,
  qcsapi_phy_mode_11ac80Cntrminus = 11,
  qcsapi_phy_mode_11ac80Edgeminus = 12,
  qcsapi_nosuch_phymode = 13 }

  This enumeration represents the current Phy Mode.
- enum qcsapi_vco_lock_detect_mode_stat {
  qcsapi_vco_lock_detect_mode_disable = 0,
  qcsapi_vco_lock_detect_mode_enable = 1 }

  This enumeration represents the vco lock detect mode status.
- enum qcsapi_mac_address_filtering {
  qcsapi_disable_mac_address_filtering = 0,
  qcsapi_accept_mac_address_unless_denied,
  qcsapi_deny_mac_address_unless_authorized,
  qcsapi_nosuch_mac_address_filtering = -1 }

  This enumeration represents the state of the MAC address filtering.
- enum qcsapi_ap_isolate_type {
  qcsapi_ap_isolate_disabled = 0,
  qcsapi_ap_isolate_enabled = 1,
  qcsapi_ap_isolation_end }

  Enumeration to represent AP isolation status.
- enum qcsapi_flash_partiton_type {
  qcsapi_image_linux_live = 0,
  qcsapi_live_image = qcsapi_image_linux_live,
  qcsapi_image_linux_safety,
  qcsapi_safety_image = qcsapi_image_linux_safety,
  qcsapi_image_uboot_live,
  qcsapi_image_uboot_safety,
  qcsapi_nosuch_partition = -1 }

  This enumeration represents the partitions supported for firmware upgrade.
- enum qcsapi_wps_param_type {
  qcsapi_wps_uuid = 0,
  qcsapi_wps_os_version,
  qcsapi_wps_device_name,
  qcsapi_wps_config_methods,
  qcsapi_wps_ap_setup_locked,
  qcsapi_wps_vendor_spec,
  qcsapi_wps_ap_pin,
  qcsapi_wps_force_broadcast_uuid,
  qcsapi_wps_ap_pin_fail_method,
  qcsapi_wps_auto_lockdown_max_retry,

qcsapi_wps_last_successful_client,
qcsapi_wps_last_successful_client_devname,
qcsapi_wps_auto_lockdown_fail_num,
qcsapi_wps_serial_number,
qcsapi_wps_manufacturer,
qcsapi_wps_model_name,
qcsapi_wps_model_number,
qcsapi_wps_pbc_in_m1,
qcsapi_wps_third_party_band,
qcsapi_wps_last_config_error,
qcsapi_wps_registrar_number,
qcsapi_wps_registrar_established,
qcsapi_wps_param_end }

Enumeration to represent WPS parameters as used by the qcsapi_wps_get_param API.

- enum qcsapi_vlan_cmd {
  e_qcsapi_vlan_add = 0x00000001,
  e_qcsapi_vlan_del = 0x00000002,
  e_qcsapi_vlan_pvid = 0x00000004,
  e_qcsapi_vlan_mode = 0x00000008,
  e_qcsapi_vlan_access = 0x00000010,
  e_qcsapi_vlan_trunk = 0x00000020,
  e_qcsapi_vlan_hybrid = 0x00000040,
  e_qcsapi_vlan_tag = 0x00000100,
  e_qcsapi_vlan_untag = 0x00000200,
  e_qcsapi_vlan_dynamic = 0x00000400,
  e_qcsapi_vlan_undynamic = 0x00000800,
  e_qcsapi_vlan_enable = 0x00001000,
  e_qcsapi_vlan_disable = 0x00002000,
  e_qcsapi_vlan_reset = 0x00004000 }

  Enumeration to represent VLAN configuration command as used by the qcsapi_wifi_vlan_config API.
- enum qcsapi_system_status {
  qcsapi_sys_status_ethernet = 0,
  qcsapi_sys_status_pcie_ep = 1,
  qcsapi_sys_status_pcie_rc = 2,
  qcsapi_sys_status_wifi = 3,
  qcsapi_sys_status_rpcd = 4,
  qcsapi_sys_status_cal_mode = 30,
  qcsapi_sys_status_completed = 31 }

  Bit number to represent system status value.
- enum qcsapi_eth_dscp_oper {
  qcsapi_eth_dscp_fill = 1,
  qcsapi_eth_dscp_poke = 2,
  qcsapi_eth_dscp_dump = 3 }

  Enumeration to represent ehternet DSCP operation command as used by the qcsapi_eth_dscp_map API.
- enum qcsapi_emac_switch {
  qcsapi_emac_switch_enable = 0,
  qcsapi_emac_switch_disable = 1 }

  Enumeration to represent EMAC switch connectivity.
- enum qcsapi_airtime_control {
  qcsapi_accum_airtime_start = 1 << 0,
  qcsapi_accum_airtime_stop = 1 << 1 }

  Enumeration to control airtime accumulation.
- enum qcsapi_br_isolate_cmd {
  e_qcsapi_br_isolate_normal = 0,
  e_qcsapi_br_isolate_vlan }

  Enumeration to represent br_isolate commands as used by the qcsapi_wifi_set_br_isolate API.

- enum **qcsapi_extender_type** {
  qcsapi_extender_role = 1,
  qcsapi_extender_mbs_best_rssi,
  qcsapi_extender_rbs_best_rssi,
  qcsapi_extender_mbs_wgt,
  qcsapi_extender_rbs_wgt,
  qcsapi_extender_verbose,
  qcsapi_extender_roaming,
  qcsapi_extender_bgscan_interval,
  qcsapi_extender_mbs_rssi_margin,
  qcsapi_extender_nosuch_param = 0 }
- enum **qcsapi_tdls_type** {
  qcsapi_tdls_over_qhop_enabled = 1,
  qcsapi_tdls_link_timeout_time,
  qcsapi_tdls_indication_window,
  qcsapi_tdls_chan_switch_mode,
  qcsapi_tdls_chan_switch_off_chan,
  qcsapi_tdls_chan_switch_off_chan_bw,
  qcsapi_tdls_discovery_interval,
  qcsapi_tdls_node_life_cycle,
  qcsapi_tdls_verbose,
  qcsapi_tdls_mode,
  qcsapi_tdls_min_rssi,
  qcsapi_tdls_link_weight,
  qcsapi_tdls_rate_weight,
  qcsapi_tdls_training_pkt_cnt,
  qcsapi_tdls_switch_ints,
  qcsapi_tdls_path_select_pps_thrshld,
  qcsapi_tdls_path_select_rate_thrshld,
  qcsapi_tdls_nosuch_param = 0 }
- enum **qcsapi_tdls_oper** {
  qcsapi_tdls_oper_discover = 1,
  qcsapi_tdls_oper_setup,
  qcsapi_tdls_oper_teardown,
  qcsapi_tdls_oper_switch_chan,
  qcsapi_tdls_nosuch_oper = 0 }
- enum **qcsapi_eth_info_result** {
  qcsapi_eth_info_connected = 0x00000001,
  qcsapi_eth_info_speed_unknown = 0x00000002,
  qcsapi_eth_info_speed_10M = 0x00000004,
  qcsapi_eth_info_speed_100M = 0x00000008,
  qcsapi_eth_info_speed_1000M = 0x00000010,
  qcsapi_eth_info_speed_10000M = 0x00000020,
  qcsapi_eth_info_duplex_full = 0x00000040,
  qcsapi_eth_info_autoneg_on = 0x00000080,
  qcsapi_eth_info_autoneg_success = 0x00000100,
  qcsapi_eth_info_unknown = 0 }
- enum **qcsapi_eth_info_type_mask** {
  qcsapi_eth_info_link_mask = qcsapi_eth_info_connected,
  qcsapi_eth_info_speed_mask,
  qcsapi_eth_info_duplex_mask = qcsapi_eth_info_duplex_full,
  qcsapi_eth_info_autoneg_mask,
  qcsapi_eth_info_all_mask }
- enum **qcsapi_eth_info_type** {
  qcsapi_eth_info_start = 1,
  qcsapi_eth_info_link = qcsapi_eth_info_start,
  qcsapi_eth_info_speed,
  qcsapi_eth_info_duplex,

qcsapi_eth_info_autoneg,
qcsapi_eth_info_all,
qcsapi_eth_nosuch_type = 0 }

- enum **qcsapi_interface_status_code** {
qcsapi_interface_status_error,
qcsapi_interface_status_disabled,
qcsapi_interface_status_up,
qcsapi_interface_status_running }
- enum qcsapi_wifi_param_type {
qcsapi_wifi_param_dtim_period = 1,
qcsapi_wifi_nosuch_parameter = 0 }

Enumeration used in the WiFi parameter set/get API.

Variables

- struct qcsapi_sample_assoc_data **__packed**

9.2.1 Detailed Description

9.2.2 Macro Definition Documentation

9.2.2.1 MAX_NUM_OF_BANDWIDTHS

```
#define MAX_NUM_OF_BANDWIDTHS 5
```

Maximum number of bandwidths + 1.

9.2.2.2 QCSAPI_MAX_MACS_IN_LIST

```
#define QCSAPI_MAX_MACS_IN_LIST 200
```

The maximum number of MAC addresses within the qcsapi_mac_list structure.

See also

qcsapi_mac_list
qcsapi_get_client_mac_list

9.2.3 Typedef Documentation

9.2.3.1 qcsapi_int_a32

```
typedef int qcsapi_int_a32[32]
```

Basic signed int array definition for internal consistency.

9.2.3.2 qcsapi_unsigned_int

```
typedef uint32_t qcsapi_unsigned_int
```

Basic unsigned int definition for internal consistency.

9.2.3.3 qcsapi_unsigned_int64

```
typedef uint64_t qcsapi_unsigned_int64
```

Basic unsigned 64-bit int definition for internal consistency.

9.2.3.4 qcsapi_mac_addr

```
typedef uint8_t qcsapi_mac_addr[MAC_ADDR_SIZE]
```

Convenience definition to represent a 6 byte MAC address.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a MAC address.

9.2.3.5 qcsapi_SSID

```
typedef char qcsapi_SSID[QCSAPI_SSID_MAXLEN]
```

Convenience definition for a string large enough for a single SSID.

This typedef has enough room for a single SSID plus the NULL terminating character.

The content within the SSID must be a string with between 1 and 32 characters. Control characters (^C, ^M, etc.) are not permitted in API calls using this type.

9.2.3.6 qcsapi_mcs_rate

```
typedef char qcsapi_mcs_rate[QCSAPI_MCS_RATE_MAXLEN]
```

QCSAPI MCS rate maximum length is distinct from MaxBitRate in TR-98. TR-98 provides for 4 characters to represent the bit rate in MBPS. QCSAPI MCS rate stores MCS rate specs - e.g. MCS0, MCS6, MCS76, etc. Provide a bit more space for future expansion. As with all QCSAPI maximum length definitions, space for the NUL ('\0') is included. So only QCSAPI_MCS_RATE_MAXLEN - 1 (7) non-NUL chars are available.

9.2.3.7 string_16

```
typedef char string_16[17]
```

Convenience definition for a string of size 16.

This type can contain a string of maximum size 16 bytes, plus the NULL terminating character.

9.2.3.8 string_32

```
typedef char string_32[33]
```

Convenience definition for a string of size 32.

This type can contain a string of maximum size 32 bytes, plus the NULL terminating character.

9.2.3.9 string_64

```
typedef char string_64[65]
```

Convenience definition for a string of size 64.

This type can contain a string of maximum size 64 bytes, plus the NULL terminating character.

9.2.3.10 string_128

```
typedef char string_128[129]
```

Convenience definition for a string of size 128.

This type can contain a string of maximum size 128 bytes, plus the NULL terminating character.

9.2.3.11 string_256

```
typedef char string_256[257]
```

Convenience definition for a string of size 256.

This type can contain a string of maximum size 256 bytes, plus the NULL terminating character.

9.2.3.12 string_512

```
typedef char string_512[513]
```

Convenience definition for a string of size 512.

This type can contain a string of maximum size 512 bytes, plus the NULL terminating character.

9.2.3.13 string_1024

```
typedef char string_1024[1025]
```

Convenience definition for a string of size 1024.

This type can contain a string of maximum size 1024 bytes, plus the NULL terminating character.

9.2.3.14 string_2048

```
typedef char string_2048[2049]
```

Convenience definition for a string of size 2048.

This type can contain a string of maximum size 2048 bytes, plus the NULL terminating character.

9.2.3.15 string_4096

```
typedef char string_4096[4097]
```

Convenience definition for a string of size 4096.

This type can contain a string of maximum size 4096 bytes, plus the NULL terminating character.

9.2.3.16 qcsapi_ap_properties

```
typedef struct qcsapi_ap_properties qcsapi_ap_properties
```

This structure represents a set of properties for a single AP.

The contents of this structure can be obtained using the function qcsapi_wifi_get_properties_AP.

This structure is used to return AP scan results.

See also

qcsapi_wifi_get_properties_AP

9.2.3.17 qcsapi_node_stats

```
typedef struct qcsapi_node_stats qcsapi_node_stats
```

This structure is used as a return parameter in the per-node association APIs associated with statistics gathering.

See also

qcsapi_wifi_get_node_stats

9.2.3.18 qcsapi_interface_stats

```
typedef struct _qcsapi_interface_stats qcsapi_interface_stats
```

This structure is used as a return parameter in the per-interface APIs associated with statistics gathering.

See also

qcsapi_get_interface_stats

9.2.3.19 qcsapi_phy_stats

```
typedef struct _qcsapi_phy_stats qcsapi_phy_stats
```

This structure is used as a return parameter in the per-interface APIs associated with PHY statistics gathering.

See also

qcsapi_get_phy_stats

9.2.3.20 qcsapi_mlme_stats

```
typedef struct _qcsapi_mlme_stats qcsapi_mlme_stats
```

This structure is used as a return parameter in the mlme statistics request functions.

See also

qcsapi_wifi_get_mlme_stats_per_association

qcsapi_wifi_get_mlme_stats_per_mac

9.2.3.21 qcsapi_mlme_stats_macs

```
typedef struct _qcsapi_mlme_stats_macs qcsapi_mlme_stats_macs
```

This structure is used as a return parameter in mlme statistics macs request function

See also

qcsapi_wifi_get_mlme_stats_macs_list

9.2.3.22 qcsapi_scs_currchan_rpt

```
typedef struct qcsapi_scs_currchan_rpt qcsapi_scs_currchan_rpt
```

This structure is used as a return parameter in the SCS API to return report for the current channel.

See also

qcsapi_wifi_get_scs_currchan_report

9.2.3.23 qcsapi_scs_ranking_rpt

```
typedef struct qcsapi_scs_ranking_rpt qcsapi_scs_ranking_rpt
```

This structure is used as a return parameter in the SCS API to return report for the all channels.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_scs_stat_report

9.2.3.24 qcsapi_scs_score_rpt

```
typedef struct qcsapi_scs_score_rpt qcsapi_scs_score_rpt
```

This structure is used as a return parameter in the SCS API to return the scores of the all channels.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_scs_score_report

9.2.3.25 qcsapi_autochan_rpt

```
typedef struct qcsapi_autochan_rpt qcsapi_autochan_rpt
```

This structure is used as a return parameter in the Auto Channel API to return report for initial channel selection.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_autochan_report

9.2.3.26 qcsapi_scs_param_rpt

```
typedef struct qcsapi_scs_param_rpt qcsapi_scs_param_rpt
```

This structure is the same as 'struct ieee80211req_scs_param_rpt', but (re)defined for convenience

9.2.3.27 qcsapi_assoc_records

```
typedef struct qcsapi_assoc_records qcsapi_assoc_records
```

Used with API 'qcsapi_wifi_get_assoc_records'

9.2.3.28 qcsapi_node_txrx_airtime

```
typedef struct _qcsapi_node_txrx_airtime qcsapi_node_txrx_airtime
```

Used with API 'qcsapi_wifi_node_get_txrx_airtime'

9.2.3.29 qcsapi_csw_record

```
typedef struct _qcsapi_csw_record qcsapi_csw_record
```

Channel switch history record

9.2.3.30 qcsapi_dscp2ac_data

```
typedef struct _qcsapi_dscp2ac_data qcsapi_dscp2ac_data
```

Data should be set to the dscp to ac mapping

9.2.3.31 qcsapi_radar_status

```
typedef struct _qcsapi_radar_status qcsapi_radar_status
```

Each channel's Radar status and detected history records

9.2.3.32 qcsapi_disconn_info

```
typedef struct _qcsapi_disconn_info qcsapi_disconn_info
```

Connection and Disconnecttion count information

9.2.3.33 qcsapi_calcmd_tx_power_rsp

```
typedef struct _qcsapi_calcmd_tx_power_rsp qcsapi_calcmd_tx_power_rsp
```

Retrieve values of tx_power on all antennas for calcmd

9.2.3.34 qcsapi_calcmd_rssi_rsp

```
typedef struct _qcsapi_calcmd_rssi_rsp qcsapi_calcmd_rssi_rsp
```

Retrieve values of rssi on all antennas for calcmd

9.2.3.35 qcsapi_measure_request_param

```
typedef union _qcsapi_measure_request_param qcsapi_measure_request_param
```

Request parameter union for 11h and 11k measurement

See also

qcsapi_wifi_get_node_param

9.2.3.36 qcsapi_measure_report_result

```
typedef union _qcsapi_measure_report_result qcsapi_measure_report_result
```

Report results for 11h and 11k measurement

See also

qcsapi_wifi_get_node_param

9.2.3.37 qcsapi_vlan_config

```
typedef struct _qcsapi_vlan_config qcsapi_vlan_config
```

per-interface VLAN configuration.
The definition must be in sync with 'struct qtn_vlan_config' in include/qtn/qtn_vlan.h

9.2.4 Enumeration Type Documentation

9.2.4.1 qcsapi_errno

```
enum qcsapi_errno
```

This enumeration represents the internal QCSAPI error return values that may be returned by various QCSAPIs. Some errors may be returned from many different APIs, whereas other errors are for only one API call.

Each error code indicates the area of the QCSAPI the code is relevant to.

To get an error string associated with the error message, use the API call qcsapi_errno_get_message.

In addition to the error codes listed in the following sections (which start at error number 1000 - `qcsapi_errno_base`), the following POSIX defined errors are used in the QCSAPI:

ERRNO value	QCSAPI Error	Description
-EFAULT	QCS API error 14: Bad address	The QCSAPI found a problem with an argument passed by reference; most likely the address was the NULL address.
-EINVAL	QCS API error 22: Invalid argument	The QCSAPI found the value of an argument is not valid. Examples are numeric value out of range (eg, WiFi channel larger than 255), or a parameter value not allowed by the WiFi standard.
-ENODEV	QCS API error 19: No such device	No such device. An operation was attempted using a device that does not exist.
-EOPNOTSUPP	QCS API error 95: Operation not supported	Operation not supported. For example, an operation limited to a WiFi device such as get 802.11 standard or get beacon type was attempted using an interface that is not a WiFi device.
-ERANGE	QCS API error 34: Parameter value out of range	This error occurs when the API accesses an element in an array using an index parameter, and the index is too large or out of range. An example is the per-association APIs.

See also

qcsapi_errno_get_message

Enumerator

qcsapi_system_not_started	This error code is returned when attempts are made to apply changes when the wireless system is not started. The most typical situation this error message is returned is when the Quantenna kernel modules have not been loaded. Many different QCSAPI function calls attempt to apply changes, and the majority of QCSAPI calls dealing with the wireless driver may return this value. `call_qcsapi` printed error message: `QCS API error 1000: System not started`
qcsapi_parameter_not_found	This error code is returned when an attempt to read in an unknown parameter via the qcsapi_config_get_parameter. **See also** qcsapi_config_get_parameter `call_qcsapi` printed error message: `QCS API error 1001: Parameter not found`
qcsapi_SSID_not_found	This error code is returned when an SSID API call is made, but the SSID referred to does not exist. The SSID may not exist due to the config file being missing, or due to the config file not containing the passed in SSID. See SSID APIs. `call_qcsapi` printed error message: `QCS API error 1002: SSID not found`

Enumerator

qcsapi_only_on_AP	This error code is returned when a QCSAPI call is attempted on an STA device, but the call only applies to the AP. This return value is used in many different QCSAPIs, across all functional areas. `call_qcsapi` printed error message: `QCS API error 1003: Operation only available on an AP`
qcsapi_only_on_STA	This error code is returned when a QCSAPI call is attempted on an AP device, but the call only applies to the STA. This return value is used in many different QCSAPIs, across all functional areas. `call_qcsapi` printed error message: `QCS API error 1004: Operation only available on a STA`
qcsapi_configuration_error	This error code is returned when the action implied by the API conflicts with the current configuration. An example is getting a list of authorized MAC addresses when MAC address filtering is not enabled. `call_qcsapi` printed error message: `QCS API error 1005: Configuration error`
qcsapi_buffer_overflow	This error code is returned when a variable length input buffer is too small for the QCSAPI result. For example, when retrieving error messages. `call_qcsapi` printed error message: `QCS API error 1006: Insufficient space in the string to receive results`
qcsapi_internal_format_error	This error code is returned when an internal error is detected when parsing config files or other data sets. `call_qcsapi` printed error message: `QCS API error 1007: Internal formatting error`
qcsapi_programming_error	This error code is returned when a system call is made in the code and it fails for some reason. `call_qcsapi` printed error message: `QCS API error 1008: Internal API programming error`
qcsapi_bringup_mode_only	This error code is returned when a QCSAPI call is made that is only supported in bringup mode. See Production Mode vs Bringup Mode `call_qcsapi` printed error message: `QCS API error 1009: Operation only available in bringup mode`
qcsapi_daemon_socket_error	This error code is returned when a socket connection to the security daemon (opened to send a command to the running daemon) fails for whatever reason. If this error is returned, one or more of the sequence of events in the QCSAPI call has failed, and the system may be in an inconsistent state. `call_qcsapi` printed error message: `QCS API error 1010: Cannot contact security manager`
qcsapi_conflicting_options	This error code is deprecated and not returned by any current API.

Enumerator

qcsapi_SSID_parameter_not_found	This error code is returned when the SSID cannot be found (when searching to see if an SSID is present). `call_qcsapi` printed error message: `QCS API error 1012: Required parameter not found in the SSID configuration block`
qcsapi_not_initialized	This error code is returned when qcsapi_init has not been called prior to invoking certain APIs (that require qcsapi_init to be called). `call_qcsapi` printed error message: `QCS API error 1013: Initialization API qcsapi_init has not been called`
qcsapi_invalid_type_image_file	This error code is returned when the flash upgrade image is not a regular file on the filesystem (eg, is a directory or device special file). `call_qcsapi` printed error message: `QCS API error 1014: Invalid file type for a flash image update file`
qcsapi_image_file_failed_chkimage	This error code is returned when the flash upgrade image fails verification checks. `call_qcsapi` printed error message: `QCS API error 1015: chkimage utility failed for the flash image update file`
qcsapi_flash_partition_not_found	This error code is returned when the flash upgrade partition is not found or is invalid. `call_qcsapi` printed error message: `QCS API error 1016: flash partition not found`
qcsapi_erase_flash_failed	This error code is returned when the command to erase the flash partition failed. `call_qcsapi` printed error message: `QCS API error 1017: failed to erase the flash memory partition`
qcsapi_copy_image_flash_failed	This error code is returned when the copy of the flash image into the flag part failed. `call_qcsapi` printed error message: `QCS API error 1018: failed to copy the new image to the flash memory partition`
qcsapi_invalid_wifi_mode	This error code is returned when a call is made into an API where the operational state of the system is not known. This is an internal error, and should never be seen in ordinary circumstances. `call_qcsapi` printed error message: `QCS API error 1019: invalid WiFi mode`
qcsapi_process_table_full	This error code is returned when the call to qcsapi_console_disconnect fails due to not enough system resources. `call_qcsapi` printed error message: `QCS API error 1020: Process table is full`
qcsapi_measurement_not_available	This error code is deprecated and not returned by any current API.
qcsapi_too_many_bssids	This error code is returned when trying to create a new BSS, but the maximum number of BSSes are already created. `call_qcsapi` printed error message: `QCS API error 1022: Maximum number of BSSIDs / VAPs exceeded`

Enumerator

qcsapi_only_on_primary_interface	This error code is returned when an operation is attempted on a non-primary interface (wifi0). This can happen for certain security settings and when performing WDS functions. `call_qcsapi` printed error message: `QCS API error 1023: Operation only available on the primary WiFi interface`
qcsapi_too_many_wds_links	This error code is returned when trying to create a new WDS link, but the maximum number of WDS links are already created. `call_qcsapi` printed error message: `QCS API error 1024: Maximum number of WDS links exceeded`
qcsapi_config_update_failed	This error code is returned when an attempt to update a config file (persistent file) fails. `call_qcsapi` printed error message: `QCS API error 1025: Failed to update persistent configuration`
qcsapi_no_network_counters	This error code is returned when the /proc/net/dev or /proc/net/packets device files are not present on the filesystem. `call_qcsapi` printed error message: `QCS API error 1026: Cannot access network counters`
qcsapi_invalid_pm_interval	This error code is returned when the PM interval passed in is invalid. That is, it is not one of the supported interval device files. `call_qcsapi` printed error message: `QCS API error 1027: Invalid performance monitoring interval`
qcsapi_only_on_wds	This error code is returned when an operation relevant only to WDS mode is attempted on a non-WDS operational mode device. `call_qcsapi` printed error message: `QCS API error 1028: Operation only available on a WDS device`
qcsapi_only_unicast_mac	This error code is returned when an multicast or broadcast MAC is used where only unicast MAC is allowed. `call_qcsapi` printed error message: `QCS API error 1029: Only unicast MAC address is allowed`
qcsapi_primary_iface_forbidden	This error code is returned when performing an invalid operation. `call_qcsapi` printed error message: `QCS API error 1030: Operation is not available on the primary interface`
qcsapi_invalid_ifname	This error code is returned when a BSS is created, but the interface name is incorrect.The BSS prefix name must be the string 'wifi'. `call_qcsapi` printed error message: `QCS API error 1031: Invalid BSS name`
qcsapi_iface_error	This error code is returned when an error happens on interface. `call_qcsapi` printed error message: `QCS API error 1032: An error happened on interface`
qcsapi_sem_error	This error code is returned when a semaphore takes too long to initialize. `call_qcsapi` printed error message: `QCS API error 1033: Semaphore initialization error`

Enumerator

qcsapi_not_supported	This error code is returned when a command is issued for a feature that is not supported in this image. `call_qcsapi` printed error message: `QCS API error 1034: Feature is not supported`
qcsapi_invalid_dfs_channel	This error code is returned when a channel as input is not a dfs channel `call_qcsapi` printed error message: `QCS API error 1035: API requires a dfs channel`
qcsapi_script_error	This error code is returned when a file can not be found. `call_qcsapi` printed error message: `QCS API error 1036: Script failed`
qcsapi_invalid_wds_peer_addr	This error code is returned when set mac address of wds peer is local address. `call_qcsapi` printed error message: `QCS API error 1037: Local Mac address can't be used as wds peer address`
qcsapi_band_not_supported	This error code is returned when band is not supported. `call_qcsapi` printed error message: `QCS API error 1038: Band is not supported`
qcsapi_region_not_supported	This error code is returned when region is not supported. `call_qcsapi` printed error message: `QCS API error 1039: Region is not supported`
qcsapi_region_database_not_found	This error code is returned when region database is not found. `call_qcsapi` printed error message: `QCS API error 1040: Region database is not found`
qcsapi_param_name_not_supported	This error code is returned when a parameter name is not supported by wireless_conf.txt. `call_qcsapi` printed error message: `QCS API error 1041: Parameter name is not supported`
qcsapi_param_value_invalid	This error code is returned when parameter value is invalid in wireless_conf.txt. `call_qcsapi` printed error message: `QCS API error 1042: Parameter value is invalid`
qcsapi_invalid_mac_addr	This error code is returned when an input MAC address is invalid `call_qcsapi` printed error message: `QCS API error 1043: Invalid MAC address`
qcsapi_option_not_supported	This error code is returned when an option is not supported. `call_qcsapi` printed error message: <QCS API error 1044: Option is not supported>
qcsapi_wps_overlap_detected	This error code is returned when a wps overlap detected `call_qcsapi` printed error message: <QCS API error 1045: WPS Overlap detected>
qcsapi_mlme_stats_not_supported	This error code is returned when a statistics module is not supported `call_qcsapi` printed error message: <QCS API error 1046: MLME statistics is not supported>

Enumerator

qcsapi_board_parameter_not_supported	This error code is returned when a board parameter requested for is not supported. `call_qcsapi` printed error message: <QCS API error 1047: Board parameter is not supported>
qcsapi_iface_invalid	This error code is returned when a QCSAPI call is attempted on an interface, but the call is not permitted. This return value is used in many different QCSAPIs, across all functional areas. `call_qcsapi` printed error message: `QCS API error 1052: Operation is not supported on this interface`
qcsapi_exceed_config_number	This error code is returned when the QCSAPI call is attempting to add too many values to multi-configuration options. For example, if more than three radius servers are added `call_qcsapi` printed error message: `QCS API error 1054: Exceeds the allowed multi-config limit`

9.2.4.2 qcsapi_led

`enum qcsapi_led`

This enumeration represents an abstract LED value.

9.2.4.3 qcsapi_auth_crypto

`enum qcsapi_auth_crypto`

This enumeration represents a set of security and authentication modes.

The security mode consists of an authentication method (eg, WPA, WPA2, EAP, etc.) and an encryption method (eg, WEP, TKIP, CCMP). These are represented in this enumeration.

See Authentication protocols and encrypyion for details of the difference between authentication and encryption.

Enumerator

qcsapi_protocol_WPA_mask	This value represents WPA v1 authentication mode.
qcsapi_protocol_11i_mask	This value represents WPA v2 authentication mode.
qcsapi_ap_PSK_authentication	This value represents preshared key authentication.
qcsapi_ap_EAP_authentication	This value represents EAP authentication.
qcsapi_ap_TKIP_encryption_mask	Thie value represents use of the TKIP cipher for encryption.
qcsapi_ap_CCMP_encryption_mask	Thie value represents use of the CCMP cipher for encryption.
qcsapi_ap_security_enabled	This value represents security is enabled on the interface.

9.2.4.4 qcsapi_gpio_config

enum qcsapi_gpio_config

This enumeration is used to represent GPIO state.

Enumerator

qcsapi_gpio_not_available	This value indicates that the GPIO isn't available for some reason.
qcsapi_gpio_input_only	Thie value indicates that the GPIO is set to input only mode.
qcsapi_gpio_output	Thie value indicates that the GPIO is set to output only.
qcsapi_nosuch_gpio_config	This is the invalid value - representing that a GPIO is not present on the platform.

9.2.4.5 qcsapi_file_path_config

enum qcsapi_file_path_config

This enumeration is used to abstract configuration file paths.

Enumerator

qcsapi_security_configuration_path	This value is used to represent the security config file path.
qcsapi_nosuch_file_path	Placeholder - invalid value.

9.2.4.6 qcsapi_wifi_mode

enum qcsapi_wifi_mode

This enumeration represents the operational mode of the device.

Enumerator

qcsapi_mode_not_defined	This value is a valid, and indicates that programs have not configured the WiFi mode.
qcsapi_access_point	The device is operating as an AP.
qcsapi_station	The device is operating as a STA.
qcsapi_wds	The device is operating in WDS mode - wireless distribution mode, or bridged mode.
qcsapi_repeater	The device is operating in repeater mode - primary interface works as a STA other interfaces work as AP
qcsapi_nosuch_mode	Invalid mode. Placeholder.

9.2.4.7 qcsapi_rate_type

enum qcsapi_rate_type

Enumeration to represent different rate sets as used in the system.

Enumerator

qcsapi_basic_rates	The set of basic rates which must be supported by all clients.
qcsapi_operational_rates	The set of actual rates in use.
qcsapi_possible_rates	The set of all supported rates on the device.
qcsapi_nosuch_rate	Placeholder - invalid.

9.2.4.8 qcsapi_mimo_type

enum qcsapi_mimo_type

Enumerator

qcsapi_mimo_ht	11n
qcsapi_mimo_vht	11ac
qcsapi_nosuch_standard	Placeholder - invalid.

9.2.4.9 qcsapi_counter_type

enum qcsapi_counter_type

See also

qcsapi_interface_get_counter
qcsapi_interface_get_counter64
qcsapi_pm_get_counter
qcsapi_wifi_get_node_counter

9.2.4.10 qcsapi_per_assoc_param

enum qcsapi_per_assoc_param

See also

qcsapi_wifi_get_node_param

9.2.4.11 qcsapi_option_type

enum qcsapi_option_type

See also

qcsapi_wifi_get_option

qcsapi_wifi_set_option

9.2.4.12 qcsapi_board_parameter_type

enum qcsapi_board_parameter_type

See also

qcsapi_get_board_parameter

9.2.4.13 qcsapi_service_name

enum qcsapi_service_name

See also

qcsapi_service_name

9.2.4.14 qcsapi_log_module_name

enum qcsapi_log_module_name

This enumeration is used to specify which module to apply log level actions to.

See also

qcsapi_set_log_level

qcsapi_get_log_level

Enumerator

QCSAPI_WPA_SUPPLICANT	get/set the log level of wpa_supplicant
QCSAPI_HOSTAPD	get/set the log level of hostapd
QCSAPI_KERNEL	get/set the log level of the Linux kernel
QCSAPI_DRIVER	get/set the log level of the driver
QCSAPI_NOSUCH_MODULE	placeholder - unknown module

9.2.4.15 qcsapi_remote_log_action

enum qcsapi_remote_log_action

This enumeration is used to specify action for streaming the logs to NPU.

See also

qcsapi_set_remote_logging

Enumerator

QCSAPI_REMOTE_LOG_ENABLE	enable the remote logging
QCSAPI_REMOTE_LOG_DISABLE	disable the remote logging
QCSAPI_NO_SUCH_ACTION	placeholder - unknown action

9.2.4.16 qcsapi_console_action

enum qcsapi_console_action

This enumeration is used to specify action for console.

See also

qcsapi_set_console

Enumerator

QCSAPI_CONSOLE_ENABLE	enable the console
QCSAPI_CONSOLE_DISABLE	disable the console
QCSAPI_NO_ACTION	placeholder - unknown action

9.2.4.17 qcsapi_bw

enum qcsapi_bw

This enumeration represents the bandwidth in use on the device.

Enumerator

qcsapi_bw_20MHz	The device is operating in 20MHz mode.

Enumerator

qcsapi_bw_40MHz	The device is operating in 40MHz mode.
qcsapi_bw_80MHz	The device is operating in 80MHz mode.
qcsapi_bw_160MHz	The device is operating in 160MHz mode.
qcsapi_nosuch_bw	Placeholder - unknown bandwidth (indicates error).

9.2.4.18 qcsapi_power_indices

`enum qcsapi_power_indices`

This enumeration represents the indices for different spatial streams (1-4) and BF cases (on/off).

See also

qcsapi_wifi_get_chan_power_table

qcsapi_wifi_set_chan_power_table

Enumerator

QCSAPI_POWER_INDEX_BFOFF_1SS	The power index for beamforming off and 1 spatial stream.
QCSAPI_POWER_INDEX_BFOFF_2SS	The power index for beamforming off and 2 spatial streams.
QCSAPI_POWER_INDEX_BFOFF_3SS	The power index for beamforming off and 3 spatial streams.
QCSAPI_POWER_INDEX_BFOFF_4SS	The power index for beamforming off and 4 spatial streams.
QCSAPI_POWER_INDEX_BFON_1SS	The power index for beamforming on and 1 spatial stream.
QCSAPI_POWER_INDEX_BFON_2SS	The power index for beamforming on and 2 spatial streams.
QCSAPI_POWER_INDEX_BFON_3SS	The power index for beamforming on and 3 spatial streams.
QCSAPI_POWER_INDEX_BFON_4SS	The power index for beamforming on and 4 spatial streams.

9.2.4.19 qcsapi_pmf

`enum qcsapi_pmf`

This enumeration represents the 802.11w / PMF capability of the VAP.

Enumerator

qcsapi_pmf_disabled	The PMF capability is disbled.
qcsapi_pmf_optional	The PMF capability is optional.
qcsapi_pmf_required	The PMF capability is required.

9.2.4.20 qcsapi_11nac_stat

enum qcsapi_11nac_stat

This enumeration represents the bandwidth in use on the device.

This enumeration is used to set the correct bandwidth.

9.2.4.21 qcsapi_pref_band

enum qcsapi_pref_band

This enumeration represents the band in use on the device.

This enumeration is used to set the preferred bandwidth.

9.2.4.22 qcsapi_rf_chip_id

enum qcsapi_rf_chip_id

This enumeration represents the Chip ID.

This enumeration is used to get the current RF Chip ID.

9.2.4.23 qcsapi_phy_mode

enum qcsapi_phy_mode

This enumeration represents the Phy Mode.

This enumeration is used to get the current RF Phy Mode.

9.2.4.24 qcsapi_vco_lock_detect_mode_stat

enum qcsapi_vco_lock_detect_mode_stat

This enumeration represents the vco lock detect mode status.

This enumeration is used to set/get vco lock detect mode stats.

9.2.4.25 qcsapi_mac_address_filtering

enum qcsapi_mac_address_filtering

This enumeration represents the state of the MAC address filtering.

MAC address filtering can be inclusive, exclusive or disabled.

Enumerator

qcsapi_disable_mac_address_filtering	MAC address filtering is fully disabled.
qcsapi_accept_mac_address_unless_denied	MAC address inclusive filtering - allow all packets unless explicitly denied in the filter list.
qcsapi_deny_mac_address_unless_authorized	MAC address exclusive filtering - deny all packets unless explicitly allowed in the filter list.
qcsapi_nosuch_mac_address_filtering	Placeholder - indicates an error.

9.2.4.26 qcsapi_ap_isolate_type

`enum qcsapi_ap_isolate_type`

Enumerator

qcsapi_ap_isolate_disabled	AP isolation is disabled. Frames between associated stations in the BSS are passed.
qcsapi_ap_isolate_enabled	AP isolation is enabled. Frames between associated stations in the BSS are blocked.
qcsapi_ap_isolation_end	Placeholder - unused.

9.2.4.27 qcsapi_flash_partiton_type

`enum qcsapi_flash_partiton_type`

This enumeration represents the partitions supported for firmware upgrade.

The two partitions used for firmware are the live and safety images. Ideally, the safety image is never touched, and is always present to allow the system to recover to a known good (factory) setting.

Enumerator

qcsapi_image_linux_live	This represents the live image partition - the partition that should be upgraded.
qcsapi_image_linux_safety	This represents the safety image partition - this should not be touched.
qcsapi_image_uboot_live	This represents the live uboot image partition - the partition that should be upgraded.
qcsapi_image_uboot_safety	This represents the safety uboot image partition - this should not be touched.
qcsapi_nosuch_partition	Placeholder - indicates an error.

9.2.4.28 qcsapi_wps_param_type

`enum qcsapi_wps_param_type`

See also

qcsapi_wps_get_param

Enumerator

qcsapi_wps_uuid	The WPS device UUID.
qcsapi_wps_os_version	The OS version the WPS device is running.
qcsapi_wps_device_name	The device name of the WPS device.
qcsapi_wps_config_methods	The supported configuration methods (eg, PBC, PIN) of the WPS device.
qcsapi_wps_ap_setup_locked	Whether the AP setup is locked or able to be reconfigured by an external registrar.
qcsapi_wps_vendor_spec	wps vendor for specific action in WPS process
qcsapi_wps_ap_pin	The label pin of the ap which is configured in the hostapd.conf
qcsapi_wps_force_broadcast_uuid	flag to force broadcast uuid
qcsapi_wps_ap_pin_fail_method	decide the action after ap pin failure occur
qcsapi_wps_auto_lockdown_max_retry	max retry count of ap pin fail in auto_lockdown mode
qcsapi_wps_last_successful_client	last successful WPS client
qcsapi_wps_last_successful_client_devname	last successful WPS client device name
qcsapi_wps_auto_lockdown_fail_num	current ap pin fail number
qcsapi_wps_serial_number	current wps serial number
qcsapi_wps_manufacturer	current wps manufacturer name
qcsapi_wps_model_name	current wps model name
qcsapi_wps_model_number	current wps model number
qcsapi_wps_pbc_in_m1	current wps "pbc in m1" setting (0 or 1)
qcsapi_wps_third_party_band	The RF band of third party
qcsapi_wps_last_config_error	Last configuration error of WPS registrar
qcsapi_wps_registrar_number	Number of entries for WPS registrar
qcsapi_wps_registrar_established	Number of estabalished WPS registrar
qcsapi_wps_param_end	Placeholder - unused.

9.2.4.29 qcsapi_vlan_cmd

```
enum qcsapi_vlan_cmd
```

See also

qcsapi_wifi_vlan_config

Enumerator

e_qcsapi_vlan_add	Set the VLAN ID of an interface
e_qcsapi_vlan_del	Clear the VLAN ID of an interface
e_qcsapi_vlan_pvid	Set an default VLAN ID of an interface
e_qcsapi_vlan_mode	Set a VLAN mode of an interface
e_qcsapi_vlan_access	Set an interface as the VLAN access mode

Enumerator

e_qcsapi_vlan_trunk	Set an interface as the VLAN trunk mode
e_qcsapi_vlan_hybrid	Set an interface as the VLAN hybrid mode
e_qcsapi_vlan_tag	Set the VLAN tag of an interface
e_qcsapi_vlan_untag	Clear the VLAN tag of an interface
e_qcsapi_vlan_dynamic	Enable 802.1X dynamic VLAN
e_qcsapi_vlan_undynamic	Disable 802.1X dynamic VLAN
e_qcsapi_vlan_enable	Enable VLAN functionality
e_qcsapi_vlan_disable	Disable VLAN functionality
e_qcsapi_vlan_reset	Reset all VLAN information

9.2.4.30 qcsapi_system_status

`enum qcsapi_system_status`

See also

qcsapi_get_system_status

Enumerator

qcsapi_sys_status_ethernet	1 means ethernet interface is up. 0 means ethernet interface is down.
qcsapi_sys_status_pcie_ep	1 means pcie module for EP is loaded correctly. 0 means pcie module for EP is failed to load.
qcsapi_sys_status_pcie_rc	1 means pcie module for RC is loaded correctly. 0 means pcie module for RC is failed to load.
qcsapi_sys_status_wifi	1 means wifi module is loaded correctly. 0 means wifi module is failed to load.
qcsapi_sys_status_rpcd	1 means Rpcd is ready. 0 Rpcd is failed to start.
qcsapi_sys_status_cal_mode	1 means device works in calstate=3 mode. 0 means device works in calstate=0 mode.
qcsapi_sys_status_completed	1 means system boot up completely. This bit **DOES NOT** means system can work correctly. It only indicates that system gets into a stage.

9.2.4.31 qcsapi_eth_dscp_oper

`enum qcsapi_eth_dscp_oper`

See also

qcsapi_eth_dscp_map

Enumerator

qcsapi_eth_dscp_fill	Setting DSCP Priority for all levels in EMAC
qcsapi_eth_dscp_poke	Setting DSCP Priority for particulat levels in EMAC
qcsapi_eth_dscp_dump	Getting DSCP Priority for all levels in EMAC

9.2.4.32 qcsapi_emac_switch

`enum qcsapi_emac_switch`

See also

qcsapi_set_emac_switch

Enumerator

qcsapi_emac_switch_enable	switch functionality is enabled
qcsapi_emac_switch_disable	switch functionality is disabled

9.2.4.33 qcsapi_airtime_control

`enum qcsapi_airtime_control`

See also

qcsapi_wifi_node_tx_airtime_accum_control

qcsapi_wifi_tx_airtime_accum_control

Enumerator

qcsapi_accum_airtime_start	start airtime accumulation
qcsapi_accum_airtime_stop	stop airtime accumulation

9.2.4.34 qcsapi_br_isolate_cmd

`enum qcsapi_br_isolate_cmd`

See also

qcsapi_wifi_set_br_isolate

Enumerator

e_qcsapi_br_isolate_normal	Enable/Disable normal bridge isolation
e_qcsapi_br_isolate_vlan	Configure VLAN bridge isolation

9.2.4.35 qcsapi_wifi_param_type

```
enum qcsapi_wifi_param_type
```

See also

qcsapi_wifi_get_parameter

qcsapi_wifi_set_parameter

9.3 API Initialization and Process Management

Unlike most APIs, the ones documented in this section cannot be called from the scripting interface, call_qcsapi.

Functions

- int qcsapi_init (void)

  Call to initialise the runtime for QCSAPI usage.
- int qcsapi_console_disconnect (void)

  Disconnect the program from the calling terminal.

9.3.1 Detailed Description

These functions are used internally, and in linked applications (using libqcsapi), in order to setup the runtime environment for further QCSAPI calls.

9.3.2 Function Documentation

9.3.2.1 qcsapi_init()

```
int qcsapi_init (
            void  )
```

This function is required to be called prior to any other QCSAPI function.

To prevent concurrency issues, a multi-threaded application should call this function prior to creating additional threads.

Returns

0 if the configuration file path was updated successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This function does not have a `call_qcsapi` equivalent call due to its nature.

9.3.2.2 qcsapi_console_disconnect()

```
int qcsapi_console_disconnect (
            void  )
```

Often a process needs to run in background. Such a process is described as a daemon process. This API will force the process into background mode and disconnect from the controlling terminal or console. When such a process is run from the command line the effect of calling this API is immediately apparent, as the command line prompt will appear and another command can then be entered.

Use this API for any process that starts as the WiFi device boots, and is expected to remain active until the device is halted or rebooted (eg, a long-running daemon process).

Returns

0 if the configuration file path was updated successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This function does not have a `call_qcsapi` equivalent call due to its nature.

9.4 System APIs

APIs to deal with general system configuration.

Functions

- int qcsapi_wifi_startprod (void)

  Start stateless board.
- int qcsapi_is_startprod_done (int *p_status)

  Get the status of the script start-prod.
- int qcsapi_system_get_time_since_start (qcsapi_unsigned_int *p_elapsed_time)

  Get the time since the system started up.
- int qcsapi_get_system_status (qcsapi_unsigned_int *p_status)

  Get system status.
- int qcsapi_get_random_seed (struct qcsapi_data_512bytes *random_buf)

  Get pseudorandom data from /dev/urandom.
- int qcsapi_set_random_seed (const struct qcsapi_data_512bytes *random_buf, const qcsapi_unsigned_int entropy)

  Feed Linux PRNG with new seed.
- int qcsapi_get_carrier_id (qcsapi_unsigned_int *p_carrier_id)

  get carrier ID
- int qcsapi_set_carrier_id (uint32_t carrier_id, uint32_t update_uboot)

  Set carrier ID.
- int qcsapi_wifi_get_spinor_jedecid (const char *ifname, unsigned int *p_jedecid)

  get spinor jedecid.
- int qcsapi_wifi_get_bb_param (const char *ifname, unsigned int *p_jedecid)

  get bb param.
- int qcsapi_wifi_set_bb_param (const char *ifname, const qcsapi_unsigned_int p_jedecid)

  set bb param.
- int qcsapi_wifi_set_optim_stats (const char *ifname, const qcsapi_unsigned_int p_jedecid)

  enable rx optim packet stats.
- int qcsapi_wifi_set_sys_time (const uint32_t timestamp)

  set system time
- int qcsapi_wifi_get_sys_time (uint32_t *timestamp)

  get system time
- int qcsapi_set_soc_mac_addr (const char *ifname, const qcsapi_mac_addr soc_mac_addr)

  Set the mac addr of the SOC board.
- int qcsapi_get_custom_value (const char *custom_key, string_128 custom_value)

  Get a custom value.
- int qcsapi_set_custom_value (const char *custom_key, const char *custom_value)

  Set a custom value.
- int qcsapi_wifi_get_vap_default_state (int *enable)

  Get default BSS state.
- int qcsapi_wifi_set_vap_default_state (const int enable)

  Set default BSS state.
- int qcsapi_wifi_get_vap_state (const char *ifname, int *enable)

  Get state for a specified BSS.
- int qcsapi_wifi_set_vap_state (const char *ifname, const int enable)

  Set state for a specified BSS.
- int qcsapi_get_ep_status (const char *ifname, qcsapi_unsigned_int *ep_status)

  This API is used to get incremented keep alive counter value.

9.4.1 Detailed Description

This section contains functions for general system configuration not suitable for other sections.

9.4.2 Function Documentation

9.4.2.1 qcsapi_wifi_startprod()

```
int qcsapi_wifi_startprod (
            void  )
```

This API call triggers initial configuration of connected stateless board after all the parameters have been set.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Warning

This API relies on the script '/scripts/start-prod' being present on the board to work.

`call_qcsapi` interface:

```
call_qcsapi startprod
```

Unless an error occurs, the output will be the string `complete`.

9.4.2.2 qcsapi_is_startprod_done()

```
int qcsapi_is_startprod_done (
            int * p_status )
```

This API call returns the status of the script start-prod.

Parameters

p_status	return parameter to contain the value indicates the status of the script start-prod, 0 if the script start-prod has not finished or not executed, and 1 if the script start-prod has finished.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi is_startprod_done
```

Unless an error occurs, the output will be "0" or "1" which means the status of the script start-prod.

9.4.2.3 qcsapi_system_get_time_since_start()

```
int qcsapi_system_get_time_since_start (
            qcsapi_unsigned_int * p_elapsed_time )
```

This function is used to determine system uptime.

Parameters

p_elapsed_time	return parameter to store the system uptime (number of seconds since boot).

Returns

0 if the configuration file path was updated successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_time_since_start
```

The output will be the time in seconds since the system started up, or an error string.

9.4.2.4 qcsapi_get_system_status()

```
int qcsapi_get_system_status (
            qcsapi_unsigned_int * p_status )
```

This function is used to get system status.

Parameters

p_status	return parameter to store the system status. It is in bit-mask format, each bit represents different status. Possiable values are defined here QCSAPI_GET_SYSTEM_STATUS

Returns

0 if get status successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_system_status
```

9.4.2.5 qcsapi_get_random_seed()

```
int qcsapi_get_random_seed (
            struct qcsapi_data_512bytes * random_buf )
```

Get pseudorandom data from /dev/urandom. It can be used to store random seed across reboots.

Parameters

random_buf	512 bytes buffer to store random data

Returns

0 if no error occurred

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_random_seed
```

The output will be 512 bytes of random data from /dev/urandom similar to `cat /dev/urandom`

9.4.2.6 qcsapi_set_random_seed()

```
int qcsapi_set_random_seed (
            const struct qcsapi_data_512bytes * random_buf,
            const qcsapi_unsigned_int entropy )
```

Add random buffer to Linux PRNG entropy pool as well as update entropy count with an estimation of added entropy.

Parameters

random_buf	512 bytes buffer of random data
entropy	added entropy estimation

Returns

0 if no error occurred

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_random_seed <random_string> <entropy_count>
```

Unless an error occurs, the output will be the string `complete`.

Note

call_qcsapi command line interface is not suitable for setting binary data and only provided as an example, a better way is to read random buffer from a file.

9.4.2.7 qcsapi_get_carrier_id()

```
int qcsapi_get_carrier_id (
            qcsapi_unsigned_int * p_carrier_id )
```

This API call is used to retrieve current carrier ID which is taking effect.

Parameters

p_carrier_id	

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_carrier_id
```

Unless an error occurs, the output will be the carrier ID.

9.4.2.8 qcsapi_set_carrier_id()

```
int qcsapi_set_carrier_id (
            uint32_t carrier_id,
            uint32_t update_uboot )
```

This API call is used to interprete the carrier ID to a set of configurations and write the setting carrier ID back to uboot according to the second parameter.

Parameters

carrier_id	
update_uboot	whether it is needed to write the carrier ID back to uboot env.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_carrier_id <carrier_id> <update_uboot>
```

Unless an error occurs, the output will be the string `complete`.

9.4.2.9 qcsapi_wifi_get_spinor_jedecid()

```
int qcsapi_wifi_get_spinor_jedecid (
            const char * ifname,
            unsigned int * p_jedecid )
```

This API get the spinor flash jedec id.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be the jedec id.

\call_qcsapi

```
call_qcsapi get_spinor_jedecid <wifi interface>
```

9.4.2.10 qcsapi_wifi_get_bb_param()

```
int qcsapi_wifi_get_bb_param (
            const char * ifname,
            unsigned int * p_jedecid )
```

This API get the bb param.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be the status of lock detect function enabled .

\call_qcsapi

```
call_qcsapi get_bb_param<wifi interface>
```

9.4.2.11 qcsapi_wifi_set_bb_param()

```
int qcsapi_wifi_set_bb_param (
            const char * ifname,
            const qcsapi_unsigned_int p_jedecid )
```

This API set the bb param.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be `1` (enabled) or `0` (disabled)

\call_qcsapi

```
call_qcsapi set_bb_param <wifi interface>
```

9.4.2.12 qcsapi_wifi_set_optim_stats()

```
int qcsapi_wifi_set_optim_stats (
            const char * ifname,
            const qcsapi_unsigned_int p_jedecid )
```

This API enable rx optim packet stats.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be `1` (enabled) or `0` (disabled)

\call_qcsapi

```
call_qcsapi set_optim_stats <wifi interface>
```

9.4.2.13 qcsapi_wifi_set_sys_time()

```
int qcsapi_wifi_set_sys_time (
            const uint32_t timestamp )
```

This API sets system time.

Parameters

timestamp	seconds since the epoch, 1970-01-01 00:00:00 +0000 (UTC)

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be `complete`.

\call_qcsapi

```
call_qcsapi set_sys_time <seconds since epoch>
```

9.4.2.14 qcsapi_wifi_get_sys_time()

```
int qcsapi_wifi_get_sys_time (
            uint32_t * timestamp )
```

This API gets system time.

Parameters

timestamp	buffer for returned timestamp

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be the current system time in seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

\call_qcsapi

```
call_qcsapi get_sys_time
```

9.4.2.15 qcsapi_set_soc_mac_addr()

```
int qcsapi_set_soc_mac_addr (
            const char * ifname,
            const qcsapi_mac_addr soc_mac_addr )
```

This API sets the SOC's mac addr of the STB to the wlan, then the mac addr will be stored for use.

Parameters

ifname	the interface to perform the action on.
soc_mac_addr	the mac addr to set

`call_qcsapi` interface:

```
call_qcsapi set_soc_macaddr <WIFI interface> <soc mac addr>
```

Unless an error occurs, the output will be the string `complete`.

9.4.2.16 qcsapi_get_custom_value()

```
int qcsapi_get_custom_value (
            const char * custom_key,
            string_128 custom_value )
```

This API returns the value of a custom key contained in a file named `/etc/custom/key`.

Note

The filename must not have the substring '..' as any part or the filename will be rejected as invalid.

Parameters

custom_key	The custom key name
custom_value	A buffer in which to store the returned value, which must be at least 129 bytes. The value will be truncated if longer than 128 characters.

Returns

0 if the custom key is valid and contians a valid value

-qcsapi_configuration_error if the custom key has not been defined

-qcsapi_configuration_error if the key includes the string ".."

-qcsapi_configuration_error if the custom key value is an empty string

`call_qcsapi` interface:

```
call_qcsapi get_custom_value <key>
```

Unless an error occurs, the output will be the custom key value.

9.4.2.17 qcsapi_set_custom_value()

```
int qcsapi_set_custom_value (
            const char * custom_key,
            const char * custom_value )
```

This API sets the value of a custom key contained in a file named `/etc/custom/`*`key`*.

Note

The filename must not have the substring '..' as any part or the filename will be rejected as invalid.

Parameters

custom_key	The custom key name
custom_value	A pointer to string to put into custom key file, which must be at most 128 bytes. Empty string could be used to delete existing key file.

Returns

0 if the custom key is valid and contians a valid value

-qcsapi_configuration_error if the custom key has not been defined

-qcsapi_configuration_error if the key includes the string ".."

-qcsapi_configuration_error if the custom key value is an empty string and the key file doesn't exist.

-qcsapi_configuration_error if the custom key value is longer than 128 bytes

`call_qcsapi` interface:

```
call_qcsapi set_custom_value <key> <value>
```

9.4.2.18 qcsapi_wifi_get_vap_default_state()

```
int qcsapi_wifi_get_vap_default_state (
            int * enable )
```

Note

This API will always return 1 currently. It is a stub only.

Parameters

enable	buffer to return the default state. 1: enabled, 0: disabled

See also

qcsapi_wifi_get_vap_state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_vap_default_state
```

Unless an error occurs, the output will be the default VAP state.

9.4.2.19 qcsapi_wifi_set_vap_default_state()

```
int qcsapi_wifi_set_vap_default_state (
            const int enable )
```

Note

This API will not currently set the default VAP state for the given API. It is a stub only.

Parameters

enable	the default BSS state - 1: enabled, 0: disabled

See also

qcsapi_wifi_set_vap_state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_vap_default_state <enable>
```

Unless an error occurs, the output will be the string `complete`.

9.4.2.20 qcsapi_wifi_get_vap_state()

```
int qcsapi_wifi_get_vap_state (
            const char * ifname,
            int * enable )
```

Note

This API will always return 1 currently. It is a stub only.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
enable	buffer to return BSS state - 1: enabled 0: disabled

See also

qcsapi_wifi_get_vap_default_state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_vap_state <wifi interface>
```

Unless an error occurs, the output will be the state for the given SSID.

9.4.2.21 qcsapi_wifi_set_vap_state()

```
int qcsapi_wifi_set_vap_state (
            const char * ifname,
            const int enable )
```

Note

This API will not currently set the VAP state for the given API. It is a stub only.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
enable	the BSS state - 1: enabled 0 disabled

See also

qcsapi_wifi_set_vap_state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_vap_state <wifi interface> <enable>
```

Unless an error occurs, the output will be the string `complete`.

9.4.2.22 qcsapi_get_ep_status()

```
int qcsapi_get_ep_status (
            const char * ifname,
            qcsapi_unsigned_int * ep_status )
```

Parameters

ifname	\wifi interface name

Returns

>= 0 incremented keep alive counter value on success, < 0 on error. Unless an error occurs, the output will be incremented keep alive counter which is maintained in driver.

`call_qcsapi` interface:

```
call_qcsapi get_ep_status <wifi interface>
```

9.5 Parameter Configuration

Functions

- int qcsapi_config_get_parameter (const char *ifname, const char *param_name, char *param_value, const size_t max_param_len)

  Get a persistent configuration parameter.
- int qcsapi_config_update_parameter (const char *ifname, const char *param_name, const char *param_value)

  Update a persistent config parameter.
- int qcsapi_config_get_ssid_parameter (const char *ifname, const char *param_name, char *param_value, const size_t max_param_len)

  Get a MBSS persistent configuration parameter.
- int qcsapi_config_update_ssid_parameter (const char *ifname, const char *param_name, const char *param_value)

  Update a MBSS persistent config parameter.

9.5.1 Detailed Description

9.5.2 Working with the Persistent Configuration

Typically a WiFi device will have a configuration that persists across a reboot or power-cycle. Examples include the WiFi Mode (AP or Station), the WiFi Channel and the Regulatory Region.
Security configuration parameters - those parameters that are handled by the Security APIs and the SSID APIs - are stored separately. Security parameters include the SSID, the WiFi Security Protocol and the Passphrase. Security parameters should not be stored in a separate persistent configuration.

Thus parameters such as the SSID name, the WiFi Security Protocol or the Passphrase should NOT be stored using the Persistent Configuration facility described below.

9.5.3 Helper Scripts and Configuration Requirements

The Update (Persistent) Configuration Parameter API will modify a parameter that has been stored in the persistent configuration. This API relies on a helper script, `update_wifi_config`, to complete the change. If this script is not present in `/scripts`, the Update Persistent Configuration Parameter API will fail. The script itself uses the Get File Path API with parameter security to locate the folder that contains the persistent configuration. The persistent configuration is then stored in `wireless_conf.txt`. Its format consists of pairs of parameter name=value, each pair separated by an ampersand ('&').

Thus the persistent configuration file `wireless_conf.txt` is stored in the same place on the device as the security configuration files hostapd.conf and wpa_supplicant.conf.
Example `wireless_conf.txt` file:
`mode=ap&bw=40&region=us&channel=44&`

All of these conditions must be met:
The `/scripts` folder has an executable script `update_wifi_config`
Folder with the persistent configuration can be located using the Get File Path API with parameter security
Persistent configuration stored in `wireless_conf.txt`
Format of `wireless_conf.txt` consists of pairs of parameter name=value, each pair separated by an ampersand ('&')

otherwise the Update Persistent) Configuration Parameter API will fail.
The display script `get_wifi_config` will display the parameter name, value pairs. Enter:
`get_wifi_config wifi0`
to display the persistent configuration.
With the example `wireless_conf.txt` above, `get_wifi_config` would display:

```
mode=ap
bw=40
region=us
channel=44
```

9.5.4 Additional Notes

The `update_wifi_config` script verifies the values for selected parameters:

Parameter	Explanation	Valid value
mode	wifi mode	ap sta
bw	Bandwidth (802.11n) Bandwidth (802.11ac)	20/40 20/40/80
channel	WiFi channel (802.11)	Range:0 - 255
pmf	PMF capability (802.11w)	0 (disabled), 1 (optional) or 2 (disabled)

Special note regarding the WiFi channel parameter check - this should be considered a sanity check, as the list of valid WiFi channels is much more restricted than the range [0 - 255].
A change to one of these persistent parameters will not take effect until the device is rebooted. Selected parameters, e.g. the channel, can be updated dynamically using other APIs.

9.5.5 Update (Persistent) Configuration Parameter

refer to functon qcsapi_config_update_parameter

See also

qcsapi_config_update_parameter

9.5.6 Function Documentation

9.5.6.1 qcsapi_config_get_parameter()

```
int qcsapi_config_get_parameter (
            const char * ifname,
            const char * param_name,
            char * param_value,
            const size_t max_param_len )
```

Get a persistent configuration parameter.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param_name	the parameter to be retrieved
param_value	a pointer to the buffer for storing the returned value
max_param_len	the size of the buffer

Returns

0 if the parameter was returned.

`-qcsapi_parameter_not_found` if the parameter was not found.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_config_param <WiFi interface> <param_name>
```

or

```
call_qcsapi get_persistent_param <WiFi interface> <parameter name>
```

The output will be the parameter value unless an error occurs or the parameter is not found.

9.5.6.2 qcsapi_config_update_parameter()

```
int qcsapi_config_update_parameter (
            const char * ifname,
            const char * param_name,
            const char * param_value )
```

Set a persistent configuration parameter. The parameter is added if it does not already exist.

All real work for this API is done in the update_wifi_config script, as explained above.

Unlike most APIs, this one will spawn a subprocess. If for any reason the process table is full, this API will fail with unpredictable results.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param_name	the parameter to be created or updated
param_value	a pointer to a buffer containing the null-terminated parameter value string

Returns

0 if the parameter was set.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi update_config_param <WiFi interface> <param_name> <value>
```

or

```
call_qcsapi update_persistent_param <WiFi interface> <param_name> <value>
```

Unless an error occurs, the output will be the string `complete`.

Examples

To set the WiFi mode to AP, enter:

```
call_qcsapi update_persistent_param wifi0 mode ap
```

To set the WiFi channel to 36 (only effective on an AP), enter:

```
call_qcsapi update_persistent_param wifi0 channel 36
```

In each case the device will need to be rebooted for the change to take effect.

9.5.6.3 qcsapi_config_get_ssid_parameter()

```
int qcsapi_config_get_ssid_parameter (
            const char * ifname,
            const char * param_name,
            char * param_value,
            const size_t max_param_len )
```

Get a MBSS persistent configuration parameter.

Parameters

ifname	wireless interface e.g wifi1
param_name	the parameter to be retrieved
param_value	a pointer to the buffer for storing the returned value
max_param_len	the size of the buffer

Returns

0 if the parameter was returned.

`-qcsapi_parameter_not_found` if the parameter was not found.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_persistent_ssid_param <WiFi interface> <param_name>
```

The output will be the parameter value unless an error occurs or the parameter is not found.

9.5.6.4 qcsapi_config_update_ssid_parameter()

```
int qcsapi_config_update_ssid_parameter (
            const char * ifname,
            const char * param_name,
            const char * param_value )
```

Set a persistent configuration parameter. The parameter is added if it does not already exist.

All real work for this API is done in the update_per_ssid_config script, as explained above.

Unlike most APIs, this one will spawn a subprocess. If for any reason the process table is full, this API will fail with unpredictable results.

Parameters

ifname	wireless interface e.g wifi1
param_name	the parameter to be created or updated
param_value	a pointer to a buffer containing the null-terminated parameter value string

Returns

0 if the parameter was set.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi update_persistenti_ssid_param <WiFi interface> <param_name>
<value>
```

Unless an error occurs, the output will be the string `complete`.

Examples

To set the WIFI1's priority to AP, enter:

```
call_qcsapi update_persistent_ssid_param wifi1 priority 3
```

In each case the device will need to be rebooted for the change to take effect.

9.6 File Path configuration

These APIs manage file paths. Each file path is the location where groups of files that the APIs work with are found. At this time the only file path configuration available is for security.

Functions

- int qcsapi_file_path_get_config (const qcsapi_file_path_config e_file_path, char *file_path, qcsapi_unsigned_int path_size)

  Get the configuration file path.
- int qcsapi_file_path_set_config (const qcsapi_file_path_config e_file_path, const char *new_path)

  Update the location associated with the configuration file path.
- int qcsapi_restore_default_config (int flag)

  Restore the default configuration files.

9.6.1 Detailed Description

Note

A number of APIs rely on the security file path setting, including all APIs documented in the Security APIs, the MAC address filtering APIs and the SSID APIs. Results from these APIs may be inconsistent or incorrect if the security file path has not been set properly.

9.6.2 Function Documentation

9.6.2.1 qcsapi_file_path_get_config()

```
int qcsapi_file_path_get_config (
            const qcsapi_file_path_config e_file_path,
            char * file_path,
            qcsapi_unsigned_int path_size )
```

Reports the location associated with the configuration file path for the security files.

Parameters

e_file_path	should be the symbolic value `qcsapi_security_configuration_path`
file_path	return parameter to contain the configuration file path to the security files.
path_size	the size of the file_path parameter above.

Returns

0 if the configuration file path was set in the file_path parameter.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_file_path security
```

The command should be entered exactly as shown above. The output is the current configured path to the security files.

9.6.2.2 qcsapi_file_path_set_config()

```
int qcsapi_file_path_set_config (
            const qcsapi_file_path_config e_file_path,
            const char * new_path )
```

Updates the location associated with the file path configuration.

Parameters

e_file_path	Use the symbolic value `qcsapi_security_configuration_path`.
new_path	A NULL terminated string for the new path to set.

Returns

0 if the configuration file path was updated successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

The presence or validity of the file path is NOT checked.

This API is only available in calibration mode (see Production mode vs calibration mode).

`call_qcsapi` interface:

```
call_qcsapi set_file_path security <new location>
```

As this is a set API, the output from call_qcsapi is complete, unless an error occurs.

Warning

Power should not be turned off to the WiFi device when calling the set file path config API or immediately afterwards. Failure to follow this restriction can cause the flash memory on the board to get corrupted. If power needs to be turned off to the WiFi device when working with this API, enter the "halt" command first and wait for the device to shut down. This API should only be called when initially configuring the board.

9.6.2.3 qcsapi_restore_default_config()

```
int qcsapi_restore_default_config (
            int flag )
```

This API call restores the board to its default (factory) configuration.

After this call, the current configurations on the board will be replaced with the default values as in the flash image.

Parameters

flag	flag specifies optional operation when restore to default configuration. It is a bitmap parameter which can be combination of Macros described below.

Returns

0 this call always succeeds.

There are Macros predefined for the parameter 'flag' as below.

```
QCSAPI_RESTORE_FG_IP               -- Restore IP address to default.
QCSAPI_RESTORE_FG_NOREBOOT -- Don't reboot device.
QCSAPI_RESTORE_FG_AP               -- Restore to AP mode.
QCSAPI_RESTORE_FG_STA              -- Restore to Station mode.
```

`call_qcsapi` interface:

```
call_qcsapi restore_default_config [0 | 1] [ap | sta] [noreboot] [ip]
```

Warning

This call will wipe out any configured parameters and replace them with the default values as per a factory configured board.

Note

This API requires the script `/scripts/restore_default_config` to be present on the filesystem.

9.7 Network Interface APIs

Network interface APIs apply to all network interfaces on the Quantenna device, including Ethernet and bridge devices as well as WiFi devices.

Functions

- int qcsapi_store_ipaddr (qcsapi_unsigned_int ipaddr, qcsapi_unsigned_int netmask)

  Stores IP address in a persistent storage.
- int qcsapi_interface_enable (const char *ifname, const int enable_flag)

  Enable or disable an interface.
- int qcsapi_interface_get_status (const char *ifname, char *interface_status)

  Get an interface status.
- int qcsapi_interface_set_ip4 (const char *ifname, const char *if_param, uint32_t if_param_val)

  Set an interface IP address or netmask.
- int qcsapi_interface_get_ip4 (const char *ifname, const char *if_param, string_64 if_param_val)

  Get an interface IP Address and netmask.
- int qcsapi_interface_get_counter (const char *ifname, qcsapi_counter_type qcsapi_counter, qcsapi_unsigned_int *p_counter_value)

  Get statistics from an interface.
- int qcsapi_interface_get_counter64 (const char *ifname, qcsapi_counter_type qcsapi_counter, uint64_t *p_counter_value)

  Get statistics from an interface.
- int qcsapi_interface_get_mac_addr (const char *ifname, qcsapi_mac_addr current_mac_addr)

  Get the MAC address of an interface.
- int qcsapi_interface_set_mac_addr (const char *ifname, const qcsapi_mac_addr interface_mac_addr)

  Set the MAC address of an interface.
- int qcsapi_pm_get_counter (const char *ifname, qcsapi_counter_type qcsapi_counter, const char *pm_interval, qcsapi_unsigned_int *p_counter_value)

  Get the Value of a Performance Monitoring Counter.
- int qcsapi_pm_get_elapsed_time (const char *pm_interval, qcsapi_unsigned_int *p_elapsed_time)

  Get the Elapsed Time in the current PM Interval.
- int qcsapi_eth_phy_power_control (int on_off, const char *interface)

  power on or power off the ethernet PHY.
- int qcsapi_get_emac_switch (char *buf)

  Get EMAC switch connectivity.
- int qcsapi_set_emac_switch (qcsapi_emac_switch value)

  Set EMAC switch connectivity.
- int qcsapi_eth_dscp_map (qcsapi_eth_dscp_oper oper, const char *eth_type, const char *level, const char *value, char *buf, const unsigned int size)

  Prioritizing Traffic in EMAC 0 and EMAC 1 using DSCP Commands.
- int qcsapi_get_eth_info (const char *ifname, const qcsapi_eth_info_type eth_info_type)

  get Ethernet interface information
- int qcsapi_get_client_mac_list (const char *ifname, int index, struct qcsapi_mac_list *mac_address_list)

  Get nodes or client MAC addresses behind the associated QTN node.
- int qcsapi_get_igmp_snooping_state (const char *ifname, uint32_t *igmp_snooping_state)

  Get IGMP snooping state.
- int qcsapi_set_igmp_snooping_state (const char *ifname, uint32_t igmp_snooping_state)

  Set IGMP snoop state.

9.7.1 Detailed Description

9.7.2 Function Documentation

9.7.2.1 qcsapi_store_ipaddr()

```
int qcsapi_store_ipaddr (
            qcsapi_unsigned_int ipaddr,
            qcsapi_unsigned_int netmask )
```

Stores IP address of the primary bridge interface in a file system. It will be used on reboot or on stateless initialization after startprod invocation.

Parameters

ipaddr	IPv4 address
netmask	Network mask

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi store_ipaddr ipaddr/netmask
```

Unless an error occurs, the output will be the string `complete`.

9.7.2.2 qcsapi_interface_enable()

```
int qcsapi_interface_enable (
            const char * ifname,
            const int enable_flag )
```

Enable (or disable) an interface. Use `qcsapi_interface_get_status` to establish whether an interface is enabled or disabled.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable_flag	if 0, disable the interface, else enable the interface.

Returns

0 if the interface was successfully enabled or disabled.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enable_interface <network interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

See QCSAPI Return Values for error codes and messages.

Examples:

To enable the ethernet interface eth1_0:

```
call_qcsapi enable_interface eth1_0 1
```

To disable the WiFi interface wifi0:

```
call_qcsapi enable_interface wifi0 0
```

See also

qcsapi_interface_get_status

9.7.2.3 qcsapi_interface_get_status()

```
int qcsapi_interface_get_status (
            const char * ifname,
            char * interface_status )
```

Determine whether an interface is enabled (up) or disabled (down).

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
interface_status	return parameter to indicate the interface status.

Returns

0 if the interface was successfully enabled or disabled.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_status <network interface>
```

Output is one of the words `up`, `disabled` or `error`; or an error message.

Examples:

To report the ethernet interface eth1_0, enter:

```
call_qcsapi get_status eth1_0
```

To report the WiFi interface status of wifi0, enter:

```
call_qcsapi get_status wifi0
```

9.7.2.4 qcsapi_interface_set_ip4()

```
int qcsapi_interface_set_ip4 (
            const char * ifname,
            const char * if_param,
            uint32_t if_param_val )
```

Setup IP address or netmask of the interfaces.

Parameters

ifname	
ipaddr|*netmask*	
ip_val|*netmask_val*	Value of ipaddress or netmask to set the interface.

Returns

0 if set the interface IP address or netmask.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_ip <network interface> { ipaddr <ip_val> | netmask <netmask>
}
```

Output is complete message or an error message.

To Set the IP address of br0 interface, enter:

```
call_qcsapi set_ip br0 ipaddr ip_val
```

9.7.2.5 qcsapi_interface_get_ip4()

```
int qcsapi_interface_get_ip4 (
            const char * ifname,
            const char * if_param,
            string_64 if_param_val )
```

Determine IP address and netmask of the interface.

Parameters

ifname	
ipaddr/netmask	return parameter to show the interface IP Aaddress or netmask.

Returns

0 if get the interface IP Address or netmask.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_ip <network interface> { ipaddr | netmask }
```

Output is IP address or netmask of the interface, or an error message.

To get netmask of br0 interface, enter:

```
call_qcsapi get_ip br0 netmask
```

To get the IP address and netmask of bridge interface br0, enter:

```
call_qcsapi get_ip br0
```

9.7.2.6 qcsapi_interface_get_counter()

```
int qcsapi_interface_get_counter (
            const char * ifname,
            qcsapi_counter_type qcsapi_counter,
            qcsapi_unsigned_int * p_counter_value )
```

This API call returns statistics for a given interface, and given counter.

Per the TR-098 standard, all counters are represented as 32-bit unsigned integers.

Note

Be aware that rollover is quite possible, especially with counters reporting the number of bytes transferred.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
qcsapi_counter	one of the enumerations from the typedef struct qcsapi_counter_type (see the following table).
p_counter_value	return parameter to contain the counter value.

Returns

0 if the value in p_counter_value is successfully filled in with the correct counter.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_counter <network device> <counter name>
```

Valid counter names are listed in the following table:

Scripting name (call_qcsapi)	C-language enum	Description
tx_bytes	qcsapi_total_bytes_sent	The number of bytes transmitted
rx_bytes	qcsapi_total_bytes_received	The number of bytes received
tx_packets	qcsapi_total_packets_sent	The number of packets transmitted
rx_packets	qcsapi_total_packets_received	The number of packets received
tx_discard	qcsapi_discard_packets_sent	The number of packets discarded on transmit
rx_discard	qcsapi_discard_packets_received	The number of packets discarded on receive
tx_errors	qcsapi_error_packets_sent	The number of packets in error on transmit
rx_errors	qcsapi_error_packets_received	The number of packets in error on receive

See also

typedef enum qcsapi_counter_type

9.7.2.7 qcsapi_interface_get_counter64()

```
int qcsapi_interface_get_counter64 (
            const char * ifname,
            qcsapi_counter_type qcsapi_counter,
            uint64_t * p_counter_value )
```

This API call returns statistics for a given interface, and given counter.

It is same with API `qcsapi_interface_get_counter`, except that it returns a 64 bit value.

See also

qcsapi_interface_get_counter

9.7.2.8 qcsapi_interface_get_mac_addr()

```
int qcsapi_interface_get_mac_addr (
            const char * ifname,
            qcsapi_mac_addr current_mac_addr )
```

Returns the MAC address of the interface. For a WiFi device operating in STA mode, this will be different from the BSSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_mac_addr	return parameter to contain the MAC address.

Returns

0 if the value in current_mac_addr is successfully filled in with the correct value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mac_addr <network device>
```

Output is the MAC address for the interface, displayed in standard format, i.e. with each byte shown as a 2 digit hexadecimal value, separated by colons, OR an error message.

Example MAC address: `02:51:55:41:00:4C`. See Format for a MAC address for details of formatting MAC addresses.

9.7.2.9 qcsapi_interface_set_mac_addr()

```
int qcsapi_interface_set_mac_addr (
            const char * ifname,
            const qcsapi_mac_addr interface_mac_addr )
```

Set the MAC address of the interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
interface_mac_addr	parameter to contain the desired MAC address.

Returns

0 if the value in interface_mac_addr is successfully set with the correct value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_mac_addr <network device> <mac address>
```

Unless an error occurs, the output will be the string `complete`.

Example MAC address: `02:51:55:41:00:4C`. See Format for a MAC address for details of formatting MAC addresses.

Note

This API needs a system reboot for the MAC address to take effect.

9.7.2.10 qcsapi_pm_get_counter()

```
int qcsapi_pm_get_counter (
            const char * ifname,
            qcsapi_counter_type qcsapi_counter,
            const char * pm_interval,
            qcsapi_unsigned_int * p_counter_value )
```

Selected counters are available as Performance Monitor (PM) counters. A PM counter is tied to a PM interval, 15 minutes or 24 hours. The PM counter records the change in the counter since the start of the current PM interval. (The Get PM Interval Elapsed Time API reports how much time has elapsed in the current PM interval.)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
qcsapi_counter	one of the enumerations from the typedef struct qcsapi_counter_type
pm_interval	the PM interval, either "15_min" or "24_hr"
p_counter_value	return parameter to contain the counter value.

Returns

0 if the value in p_counter_value is successfully filled in with the correct counter.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi get_pm_counter <network device> <counter name> <PM interval>
```

See QCSAPI Return Values for error codes and messages.

9.7.2.11 qcsapi_pm_get_elapsed_time()

```
int qcsapi_pm_get_elapsed_time (
            const char * pm_interval,
            qcsapi_unsigned_int * p_elapsed_time )
```

Returns the amount of time in seconds that has elapsed since the start of the referenced PM interval. PM Intervals last either 15 minutes (900 seconds) or 24 hours (86400 seconds).

Parameters

pm_interval	the PM interval, either "15_min" or "24_hr"
p_elapsed_time	return parameter to contain the elapsed time in seconds.

Returns

0 if the value in p_elapsed_time is successfully filled in with the elapsed time.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi get_pm_elapsed_time <PM interval>
```

See QCSAPI Return Values for error codes and messages.

9.7.2.12 qcsapi_eth_phy_power_control()

```
int qcsapi_eth_phy_power_control (
            int on_off,
            const char * interface )
```

Power off / on eth PHY. Use `qcsapi_eth_phy_power_control` to establish power on or off the eth PHY.

Parameters

on_off	if 1, power off the PHY, else power on the PHY.
interface	the interface name of the eth device

Returns

0 if the eth PHY was successfully power off or power on.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi eth_phy_power_off <network interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

See QCSAPI Return Values for error codes and messages.

Examples:

To power off the PHY of interface eth1_0:

```
call_qcsapi eth_phy_power_off eth1_0 1
```

9.7.2.13 qcsapi_get_emac_switch()

```
int qcsapi_get_emac_switch (
            char * buf )
```

This function determines if the EMACs on the board have switch functionality enabled. That is, whether traffic can be switched between devices connected directly to each port directly.

Parameters

buf	return value of the EMAC switch connectivity. 1 - switch functionality is enabled; 0 - switch functionality is disabled.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_emac_switch
```

Unless an error occurs, the output will be the value of communications between EMAC 0 and EMAC 1

See QCSAPI Return Values for error codes and messages.

9.7.2.14 qcsapi_set_emac_switch()

```
int qcsapi_set_emac_switch (
            qcsapi_emac_switch value )
```

This function sets EMAC switch connectivity.

Parameters

value	if 0, enabling emac switch functionality, else disabling emac switch functionality.

Returns

0 if the emac switch functionality successfully disabled or enabled.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_emac_switch <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

See QCSAPI Return Values for error codes and messages.

Examples:

To disable emac switch functionality:

```
call_qcsapi set_emac_switch 1
```

9.7.2.15 qcsapi_eth_dscp_map()

```
int qcsapi_eth_dscp_map (
            qcsapi_eth_dscp_oper oper,
            const char * eth_type,
            const char * level,
            const char * value,
            char * buf,
            const unsigned int size )
```

To set and get DSCP priority values in EMAC0 and EMAC1

Parameters

eth_type	type of the ethernet device. It should be emac0 or emac1
level	the dscp level, level range should not be negative and less than 64
value	the dscp priority value, value range should not be negative and less than 16
buf	a pointer to the buffer for storing the returned value
size	buf variable size

Returns

0 if the command succeeded.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi eth_dscp_map <fill/poke/dump> <eth_type> [level] [value]
```

See QCSAPI Return Values for error codes and messages.

9.7.2.16 qcsapi_get_eth_info()

```
int qcsapi_get_eth_info (
            const char * ifname,
            const qcsapi_eth_info_type eth_info_type )
```

This API gets Ethernet interface information, including link/speed/duplex/auto-negotiation.

Parameters

ifname	Ehternet interface name - e.g. eth1_0
eth_info_type	"link", "speed", "duplex" or "autoneg"

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

\call_qcsapi

```
call_qcsapi get_eth_info <Ethernet interface> { link | speed | duplex | autoneg
}
```

9.7.2.17 qcsapi_get_client_mac_list()

```
int qcsapi_get_client_mac_list (
            const char * ifname,
```

```
int index,
struct qcsapi_mac_list * mac_address_list )
```

For a Quantenna device running in four address mode, obtain the list of all device MAC addresses behind the bridge. This is possible due to the use of four address mode headers on Quantenna->Quantenna devices.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
index	index of the associated node.
mac_address_list	buffer to store the returned MAC addresses.

Returns

>=0 on success, <0 on error.

The return buffer will contain a maximum of QCSAPI_MAX_MACS_IN_LIST MAC addresses as per the struct qcsapi_mac_list, along with the number of entries contained in the array. The flags field of the MAC address list structure indicates if the results are truncated.

`call_qcsapi` interface:

```
call_qcsapi get_client_mac_list <WiFi interface> <index>
```

Unless an error occurs, the output will contain either a single MAC address (for non-Quantenna three address nodes), or for four address capable nodes, output such as below:

```
quantenna # call_qcsapi get_client_mac_list wifi0 24
Node supports 4 address
        F0:1F:AF:2F:CE:B2
        00:26:86:F0:3A:36
        00:26:86:F0:39:62
        00:19:E3:35:71:3D
quantenna # show_assoc 24
MAC               Idx  AID   Type Mode   Vendor   BW   Assoc   Auth   BA State   TDLS State        VAP
      PowerSaveSchemes
00:26:86:f0:3a:36   24   12    sta   ac      qtn   80     525      1   00430043         none      wifi0
                0
```

In the above example, the node with index 24 has four MAC addresses behind the wireless. Two are Ethernet based clients, the other two are the WiFi and Bridge MAC address of the Quantenna STA itself.

See also

qcsapi_mac_list

QCSAPI_MAX_MACS_IN_LIST

9.7.2.18 qcsapi_get_igmp_snooping_state()

```
int qcsapi_get_igmp_snooping_state (
            const char * ifname,
            uint32_t * igmp_snooping_state )
```

Get the IGMP snooping state.

Parameters

ifname	bridge interface (e.g.br0)
igmp_snooping_state	buffer to receive the returned state 1:enable 0:disabled

Returns

0 on success or negative value on error.

`call_qcsapi` interface:

```
call_qcsapi get_igmp_snooping_state <bridge interface> <igmp_snooping_↩
state>
```

9.7.2.19 qcsapi_set_igmp_snooping_state()

```
int qcsapi_set_igmp_snooping_state (
            const char * ifname,
            uint32_t igmp_snooping_state )
```

Set IGMP snooping state. IGMP snooping is enabled by default, but can be disabled if it is to be handled by an external process.

Parameters

ifname	bridge interface (e.g. br0)
igmp_snooping_state	1 to enable or 0 to disable

Returns

0 on success or negative values on error.

`call_qcsapi` interface:

```
call_qcsapi set_igmp_snooping_state <bridge interface> { 1 | 0 }
```

9.8 WiFi APIs

Wifi APIs require the interface to be a Wireless Extension (WE) device, and return Not Supported for other network interfaces.
Special note regarding the call_qcsapi interface: the default WiFi interface name is wifi0, and this interface is used to describe all interfaces to the WiFi APIs thru call_qcsapi. If a different WiFi interface name is present, that name should be used to access the WiFi APIs.

Macros

- #define **QCSAPI_CHAN_PRI_INACTIVE_AUTOCHAN** 0x1

Functions

- int qcsapi_wifi_get_mode (const char *ifname, qcsapi_wifi_mode *p_wifi_mode)

  Get the WiFi mode of the interface.
- int qcsapi_wifi_set_mode (const char *ifname, const qcsapi_wifi_mode new_wifi_mode)

  Set the WiFi mode of the interface.
- int qcsapi_wifi_get_phy_mode (const char *ifname, char *p_wifi_phy_mode)

  Get the current mode of the WLAN interface.
- int qcsapi_wifi_set_phy_mode (const char *ifname, const char *new_phy_mode)

  Set the phy mode of the WLAN interface.
- int qcsapi_wifi_reload_in_mode (const char *ifname, const qcsapi_wifi_mode new_wifi_mode)

  Reload the interface to change its mode (AP->STA or STA->AP).
- int qcsapi_wifi_rfenable (const qcsapi_unsigned_int onoff)

  Enable or disable Radio(RF).
- int qcsapi_wifi_rfstatus (qcsapi_unsigned_int *rfstatus)

  Get the current radio status.
- int qcsapi_wifi_get_bw (const char *ifname, qcsapi_unsigned_int *p_bw)

  Get the WiFi interface bandwidth (20MHz or 40MHz).
- int qcsapi_wifi_set_bw (const char *ifname, const qcsapi_unsigned_int bw)

  Set the WiFi interface bandwidth.
- int qcsapi_wifi_get_24g_bw (const char *ifname, qcsapi_unsigned_int *p_bw)

  Get the WiFi 2.4Ghz interface bandwidth (20MHz or 40MHz).
- int qcsapi_wifi_set_24g_bw (const char *ifname, const qcsapi_unsigned_int bw)

  Set the WiFi interface bandwidth.
- int qcsapi_wifi_set_vht (const char *ifname, const qcsapi_unsigned_int the_vht)

  set vht
- int qcsapi_wifi_get_vht (const char *ifname, qcsapi_unsigned_int *vht)

  get vht
- int qcsapi_wifi_get_channel (const char *ifname, qcsapi_unsigned_int *p_current_channel)

  Get the current channel.
- int qcsapi_wifi_set_channel (const char *ifname, const qcsapi_unsigned_int new_channel)

  Set the current channel.
- int qcsapi_wifi_get_chan_pri_inactive (const char *ifname, struct qcsapi_data_256bytes *buffer)

  Get the inactive channel list.
- int qcsapi_wifi_set_chan_pri_inactive (const char *ifname, const qcsapi_unsigned_int channel, const qcsapi_unsigned_int inactive)

  Set the channel inactive flag of primary channel.

- int qcsapi_wifi_set_chan_pri_inactive_ext (const char *ifname, const qcsapi_unsigned_int channel, const qcsapi_unsigned_int inactive, const uint32_t option_flags)

  Set the channel inactive flag of primary channel.
- int qcsapi_wifi_chan_control (const char *ifname, const struct qcsapi_data_256bytes *chans, const uint32_t cnt, const uint8_t flag)

  Set the channel disabled/enabled.
- int qcsapi_wifi_get_chan_disabled (const char *ifname, struct qcsapi_data_256bytes *p_chans, uint8_t *p_cnt)

  Get the disabled channel list.
- int qcsapi_wifi_get_supported_freq_bands (const char *ifname, string_32 p_bands)

  Get supported frequency bands.
- int qcsapi_wifi_get_beacon_interval (const char *ifname, qcsapi_unsigned_int *p_current_intval)

  Get the beacon interval.
- int qcsapi_wifi_set_beacon_interval (const char *ifname, const qcsapi_unsigned_int new_intval)

  Set the beacon interval.
- int qcsapi_wifi_get_dtim (const char *ifname, qcsapi_unsigned_int *p_dtim)

  Get the DTIM interval.
- int qcsapi_wifi_set_dtim (const char *ifname, const qcsapi_unsigned_int new_dtim)

  Set the DTIM interval.
- int qcsapi_wifi_get_assoc_limit (const char *ifname, qcsapi_unsigned_int *p_assoc_limit)

  Get association limit.
- int qcsapi_wifi_get_bss_assoc_limit (qcsapi_unsigned_int group, qcsapi_unsigned_int *p_assoc_limit)

  Get VAP logical group's association limit.
- int qcsapi_wifi_set_assoc_limit (const char *ifname, const qcsapi_unsigned_int new_assoc_limit)

  Set association limit.
- int qcsapi_wifi_set_bss_assoc_limit (const qcsapi_unsigned_int group, const qcsapi_unsigned_int limit)

  Set the maximum number of associations allowed per logical group of VAPs.
- int qcsapi_wifi_set_SSID_group_id (const char *ifname, const qcsapi_unsigned_int group)

  Assign VAP (SSID) to a logical group.
- int qcsapi_wifi_get_SSID_group_id (const char *ifname, qcsapi_unsigned_int *p_group)

  Get VAP's (SSID) logical group id.
- int qcsapi_wifi_set_SSID_assoc_reserve (const qcsapi_unsigned_int group, const qcsapi_unsigned_int value)

  Reserve associations for a logical group.
- int qcsapi_wifi_get_SSID_assoc_reserve (qcsapi_unsigned_int group, qcsapi_unsigned_int *p_value)

  Get number of associations reserved for a logical group.
- int qcsapi_wifi_get_BSSID (const char *ifname, qcsapi_mac_addr current_BSSID)

  Get current BSSID.
- int qcsapi_wifi_get_config_BSSID (const char *ifname, qcsapi_mac_addr config_BSSID)

  Get configured BSSID.
- int qcsapi_wifi_ssid_get_bssid (const char *ifname, const qcsapi_SSID ssid_str, qcsapi_mac_addr bssid)

  Get BSSID for a SSID.
- int qcsapi_wifi_ssid_set_bssid (const char *ifname, const qcsapi_SSID ssid_str, const qcsapi_mac_addr bssid)

  Set/Unset configured BSSID for a SSID.
- int qcsapi_wifi_get_SSID (const char *ifname, qcsapi_SSID SSID_str)

  Get the WiFi interface SSID.
- int qcsapi_wifi_set_SSID (const char *ifname, const qcsapi_SSID SSID_str)

  Set the WiFi interface SSID.
- int qcsapi_wifi_get_IEEE_802_11_standard (const char *ifname, char *IEEE_802_11_standard)

  Get the IEEE 802.11 PHY mode being used on the interface.

- int qcsapi_wifi_get_list_channels (const char *ifname, string_1024 list_of_channels)

  Get the list of available channels for an interface.
- int qcsapi_wifi_get_supp_chans (const char *ifname, qcsapi_mac_addr mac_addr, string_1024 list_of_channels)

  Get the list of supported channels for a STA.
- int qcsapi_wifi_get_mode_switch (uint8_t *p_wifi_mode_switch_setting)

  Get WiFi mode switch setting.
- int qcsapi_wifi_disassociate (const char *ifname)

  Force a disassociate on the STA.
- int qcsapi_wifi_disassociate_sta (const char *ifname, qcsapi_mac_addr mac)

  Force a disassociate on the AP. Station to disassociate identified by MAC address.
- int qcsapi_wifi_reassociate (const char *ifname)

  Force reassociation.
- int qcsapi_wifi_get_disconn_info (const char *ifname, qcsapi_disconn_info *disconn_info)

  Get information of disconnection count and time since device boot up.
- int qcsapi_wifi_disable_wps (const char *ifname, int disable_wps)

  Enable/Disable WPS.
- int qcsapi_wifi_associate (const char *ifname, const qcsapi_SSID join_ssid)

  Associate a STA to a network.
- int qcsapi_wifi_start_cca (const char *ifname, int channel, int duration)

  Trigger CCA (Clear Channel Assessment) measurement.
- int qcsapi_wifi_get_noise (const char *ifname, int *p_noise)

  Get the noise figure for the current channel.
- int qcsapi_wifi_get_rssi_by_chain (const char *ifname, int rf_chain, int *p_rssi)

  Get the RSSI measurement per RF chain.
- int qcsapi_wifi_get_avg_snr (const char *ifname, int *p_snr)

  Get the average SNR of the interface.
- int qcsapi_get_primary_interface (char *ifname, size_t maxlen)

  Get the primary wireless interface.
- int qcsapi_get_interface_by_index (unsigned int if_index, char *ifname, size_t maxlen)

  Get a wireless interface by index.
- int qcsapi_wifi_set_wifi_macaddr (const qcsapi_mac_addr new_mac_addr)

  Set the primary WiFi interface MAC address.
- int qcsapi_interface_get_BSSID (const char *ifname, qcsapi_mac_addr current_BSSID)

  Get the BSSID.
- int qcsapi_wifi_get_rates (const char *ifname, qcsapi_rate_type rate_type, string_2048 supported_rates)

  Get the list of supported rates on the given interface.
- int qcsapi_wifi_set_rates (const char *ifname, qcsapi_rate_type rate_type, const string_256 current_rates, int num_rates)

  Set rates from an MBPS list (unsupported)
- int qcsapi_get_max_bitrate (const char *ifname, char *max_bitrate, const int max_str_len)

  Get the maximum upstream and downstream bit rate available to this connection in Mbps.
- int qcsapi_set_max_bitrate (const char *ifname, const char *max_bitrate)

  Set the maximum upstream and downstream bit rate available to this connection in Mbps.
- int qcsapi_wifi_qos_get_param (const char *ifname, int the_queue, int the_param, int ap_bss_flag, int *p_value)

  Get a Quality of Service (QOS) Parameter.
- int qcsapi_wifi_qos_set_param (const char *ifname, int the_queue, int the_param, int ap_bss_flag, int value)

  Set a Quality of Service (QOS) Parameter.
- int qcsapi_wifi_get_wmm_ac_map (const char *ifname, string_64 mapping_table)

  Get the mapping table from TOS/DSCP or IEEE802.1p priority to WMM AC index.

- int qcsapi_wifi_set_wmm_ac_map (const char *ifname, int user_prio, int ac_index)

  Set one mapping table item from TOS/DSCP or IEEE802.1p priority to WMM AC index.
- int qcsapi_wifi_get_dscp_8021p_map (const char *ifname, string_64 mapping_table)

  Get the mapping table from IP DSCP to IEEE802.1p priority.
- int qcsapi_wifi_get_dscp_ac_map (const char *ifname, struct qcsapi_data_64bytes *mapping_table)

  Get IP DSCP to WMM AC mapping table entries.
- int qcsapi_wifi_set_dscp_8021p_map (const char *ifname, const char *ip_dscp_list, uint8_t dot1p_up)

  Set one mapping table item from IP DSCP to IEEE802.1p user priority.
- int qcsapi_wifi_set_dscp_ac_map (const char *ifname, const struct qcsapi_data_64bytes *dscp_list, uint8_t dscp_list_len, uint8_t ac)

  Set one mapping table item from IP DSCP to WMM AC priority.
- int qcsapi_wifi_set_qos_map (const char *ifname, const char *qos_map_str)

  Set QoS Map Set.
- int qcsapi_wifi_del_qos_map (const char *ifname)

  Remove QoS Map Set.
- int qcsapi_wifi_get_qos_map (const char *ifname, string_256 value)

  Get QoS Map Set.
- int qcsapi_wifi_send_qos_map_conf (const char *ifname, const qcsapi_mac_addr sta_mac_addr)

  Send QoS Map Configure action frame to a station.
- int qcsapi_wifi_get_dscp_tid_map (const char *ifname, struct qcsapi_data_64bytes *dscp2tid_ptr)

  Get DSCP to TID mapping.
- int qcsapi_wifi_get_priority (const char *ifname, uint8_t *p_priority)

  Get the interface priority.
- int qcsapi_wifi_set_priority (const char *ifname, uint8_t priority)

  Set the interface priority.
- int qcsapi_wifi_get_airfair (const char *ifname, uint8_t *p_airfair)

  Get the airtime fairness status.
- int qcsapi_wifi_set_airfair (const char *ifname, uint8_t airfair)

  Set airtime fairness.
- int qcsapi_wifi_get_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, int *p_tx_power)

  Get the transmit power for the current bandwidth with beamforming off.
- int qcsapi_wifi_set_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, const int tx_power)

  Set the transmit power for the current bandwidth.
- int qcsapi_wifi_get_bw_power (const char *ifname, const qcsapi_unsigned_int the_channel, int *p_power_20M, int *p_power_40M, int *p_power_80M)

  Get the transmit powers for 20/40/80MHz bandwidths with beamforming off.
- int qcsapi_wifi_set_bw_power (const char *ifname, const qcsapi_unsigned_int the_channel, const int power_20M, const int power_40M, const int power_80M)

  Set the transmit power for 20/40/80MHz bandwidths.
- int qcsapi_wifi_get_bf_power (const char *ifname, const qcsapi_unsigned_int the_channel, const qcsapi_unsigned_int number_ss, int *p_power_20M, int *p_power_40M, int *p_power_80M)

  Get the transmit powers for 20/40/80MHz bandwidths with beamforming on.
- int qcsapi_wifi_set_bf_power (const char *ifname, const qcsapi_unsigned_int the_channel, const qcsapi_unsigned_int number_ss, const int power_20M, const int power_40M, const int power_80M)

  Set the transmit power for 20/40/80MHz bandwidths.
- int qcsapi_wifi_get_tx_power_ext (const char *ifname, const qcsapi_unsigned_int the_channel, const qcsapi_unsigned_int bf_on, const qcsapi_unsigned_int number_ss, int *p_power_20M, int *p_power_40M, int *p_power_80M)

  Get the transmit powers for 20/40/80MHz bandwidths.

- int qcsapi_wifi_set_tx_power_ext (const char *ifname, const qcsapi_unsigned_int the_channel, const qcsapi_unsigned_int bf_on, const qcsapi_unsigned_int number_ss, const int power_20M, const int power_40M, const int power_80M)

  Set the transmit power for 20/40/80MHz bandwidths.
- int qcsapi_wifi_get_chan_power_table (const char *ifname, qcsapi_channel_power_table *chan_power_table)

  Get all the transmit powers for a single channel.
- int qcsapi_wifi_set_chan_power_table (const char *ifname, qcsapi_channel_power_table *chan_power_table)

  Set all the transmit powers for a single channel.
- int qcsapi_wifi_get_power_selection (qcsapi_unsigned_int *p_power_selection)

  Get the current mode for selecting power table.
- int qcsapi_wifi_set_power_selection (const qcsapi_unsigned_int power_selection)

  Set the mode for selecting power table.
- int qcsapi_wifi_get_carrier_interference (const char *ifname, int *ci)

  Get Carrier/Interference.
- int qcsapi_wifi_get_congestion_index (const char *ifname, int *ci)

  Get congestion index.
- int qcsapi_wifi_get_supported_tx_power_levels (const char *ifname, string_128 available_percentages)

  Get the Supported TX Power Levels (as a list of percentages of the maximum allowed)
- int qcsapi_wifi_get_current_tx_power_level (const char *ifname, uint32_t *p_current_percentage)

  Get the Current TX Power Level (as a percentage of the maximum allowed)
- int qcsapi_wifi_set_current_tx_power_level (const char *ifname, uint32_t txpower_percentage)

  Set TX Power Level (as a percentage of the maximum allowed)
- int qcsapi_wifi_set_power_constraint (const char *ifname, uint32_t pwr_constraint)

  Set power constraint of current channel.
- int qcsapi_wifi_get_power_constraint (const char *ifname, uint32_t *p_pwr_constraint)

  Get power constraint of current channel.
- int qcsapi_wifi_set_tpc_interval (const char *ifname, int32_t tpc_interval)

  Set tpc interval if 802_11h is enabled.
- int qcsapi_wifi_get_tpc_interval (const char *ifname, uint32_t *p_tpc_interval)

  Get tpc interval if 802_11h is enabled.
- int qcsapi_wifi_get_assoc_records (const char *ifname, int reset, qcsapi_assoc_records *records)

  Get the record of nodes that have associated with the device.
- int qcsapi_wifi_get_ap_isolate (const char *ifname, int *p_ap_isolate)

  Get the global AP isolation setting.
- int qcsapi_wifi_set_ap_isolate (const char *ifname, const int new_ap_isolate)

  Set the global AP isolation for all WiFi interfaces.
- int qcsapi_wifi_get_intra_bss_isolate (const char *ifname, qcsapi_unsigned_int *p_ap_isolate)

  Get the current intra-BSS isolation setting.
- int qcsapi_wifi_set_intra_bss_isolate (const char *ifname, const qcsapi_unsigned_int new_ap_isolate)

  Enable or disable intra-BSS isolation per BSS.
- int qcsapi_wifi_get_bss_isolate (const char *ifname, qcsapi_unsigned_int *p_ap_isolate)

  Get the current inter-BSS isolation setting.
- int qcsapi_wifi_set_bss_isolate (const char *ifname, const qcsapi_unsigned_int new_ap_isolate)

  Enable or disable inter-BSS isolation.
- int qcsapi_wifi_disable_dfs_channels (const char *ifname, int disable_dfs, int channel)

  Disable DFS channels.
- int qcsapi_wifi_is_ready (qcsapi_unsigned_int *p_value)

  Get WiFi ready state.
- int qcsapi_wifi_test_traffic (const char *ifname, uint32_t period)

Start/Stop test traffic.

- int qcsapi_wifi_add_multicast (qcsapi_unsigned_int ipaddr, qcsapi_mac_addr mac)

  Add a multicast entry.
- int qcsapi_wifi_del_multicast (qcsapi_unsigned_int ipaddr, qcsapi_mac_addr mac)

  Remove a multicast entry.
- int qcsapi_wifi_get_multicast_list (char *buf, int buflen)

  Display multicast entries.
- int qcsapi_wifi_add_ipff (qcsapi_unsigned_int ipaddr)

  Add a multicast IP address to the flood-forwarding table.
- int qcsapi_wifi_del_ipff (qcsapi_unsigned_int ipaddr)

  Remove a multicast IP address from the flood-forwarding table.
- int qcsapi_wifi_get_ipff (char *buf, int buflen)

  Display the contents of the flood-forwarding table.
- int qcsapi_wifi_get_rts_threshold (const char *ifname, qcsapi_unsigned_int *rts_threshold)

  Get RTS threshold.
- int qcsapi_wifi_set_rts_threshold (const char *ifname, qcsapi_unsigned_int rts_threshold)

  Set RTS threshold.
- int qcsapi_wifi_set_nss_cap (const char *ifname, const qcsapi_mimo_type modulation, const qcsapi_unsigned_int nss)

  set nss cap
- int qcsapi_wifi_get_nss_cap (const char *ifname, const qcsapi_mimo_type modulation, qcsapi_unsigned_int *nss)

  get nss cap
- int qcsapi_wifi_get_tx_amsdu (const char *ifname, int *enable)

  get A-MSDU status for VAP
- int qcsapi_wifi_set_tx_amsdu (const char *ifname, int enable)

  enable/disable A-MSDU for VAP
- int qcsapi_wifi_get_disassoc_reason (const char *ifname, qcsapi_unsigned_int *reason)

  get disassoc reason code
- int qcsapi_wifi_block_bss (const char *ifname, const qcsapi_unsigned_int flag)

  Block or unblock all association request for one specified BSS.
- int qcsapi_wifi_get_block_bss (const char *ifname, unsigned int *pvalue)

  Get status whether association requests for one specified BSS are blocked or not.
- int qcsapi_wifi_verify_repeater_mode (void)

  Check if device is in repeater mode.
- int qcsapi_wifi_set_ap_interface_name (const char *ifname)

  Configure the AP interface name.
- int qcsapi_wifi_get_ap_interface_name (char *ifname)

  Get the AP interface name.
- int qcsapi_wifi_set_pref_band (const char *ifname, const qcsapi_unsigned_int pref_band)

  set preferred band 2.4/5G
- int qcsapi_wifi_get_pref_band (const char *ifname, qcsapi_unsigned_int *pref_band)

  get preferred band 2.4/5G
- int qcsapi_wifi_set_txba_disable (const char *ifname, const qcsapi_unsigned_int value)

  Set TX BA disable configuration for an SSID.
- int qcsapi_wifi_get_txba_disable (const char *ifname, qcsapi_unsigned_int *value)

  Get TX BA disable state for an SSID.
- int qcsapi_wifi_set_rxba_decline (const char *ifname, const qcsapi_unsigned_int value)

  Set RX BA decline configuration for an SSID.
- int qcsapi_wifi_get_rxba_decline (const char *ifname, qcsapi_unsigned_int *value)

  Get RX BA decline state for an SSID.

- int qcsapi_wifi_set_txburst (const char *ifname, const qcsapi_unsigned_int enable)

  Set Tx burst state for a BSS.
- int qcsapi_wifi_get_txburst (const char *ifname, qcsapi_unsigned_int *enable)

  Get Tx burst state for a BSS.
- int qcsapi_wifi_get_sec_chan (const char *ifname, int chan, int *p_sec_chan)

  get secondary channel of the assigned channel
- int qcsapi_wifi_set_sec_chan (const char *ifname, int chan, int offset)

  set secondary channel of the assigned channel
- int qcsapi_wifi_node_tx_airtime_accum_control (const char *ifname, const uint32_t node_index, qcsapi_airtime_control control)

  Control per-node TX airtime accumulation start and stop.
- int qcsapi_wifi_tx_airtime_accum_control (const char *ifname, qcsapi_airtime_control control)

  Control TX airtime accumulation start and stop.
- int qcsapi_wifi_node_get_txrx_airtime (const char *ifname, const uint32_t node_index, qcsapi_node_txrx_airtime *node_txrx_airtime)

  Get per-node TX and RX airtime stats.
- int qcsapi_wifi_get_txrx_airtime (const char *ifname, string_4096 buffer)

  Get TX and RX airtime stats.
- int qcsapi_wifi_set_max_bcast_pps (const char *ifname, const qcsapi_unsigned_int max_bcast_pps)

  Set the limit for maximum broadcast packets allowed per second.
- int qcsapi_wifi_is_weather_channel (const char *ifname, const uint16_t channel)

  Check if the channel provided is a weather one.
- int qcsapi_wifi_get_tx_max_amsdu (const char *ifname, int *max_len)

  Get the configured maximum A-MSDU size.
- int qcsapi_wifi_set_tx_max_amsdu (const char *ifname, int max_len)

  Set the maximum A-MSDU size.

9.8.1 Detailed Description

9.8.2 Multiple Basic Service Sets (MBSSID)

One can create multiple Basic Service Sets (BSSIDs) on a device initially configured as an access point (AP). This capability is not available on a device configured as a STA. The first step in creating an additional BSSID is to create the wireless interface device for that BSSID. Use the Set (WiFi) Mode API for this purpose.

9.8.3 Primary versus Secondary Wireless Interface

A number of APIs change or report properties of the WiFi radio. An example is the current WiFi channel. A change in the WiFi channel affects all wireless interfaces. To prevent unexpected side effects, the concept of the Primary Wireless Interface is introduced. Selected WiFi APIs only work on the primary interface. An example is the Set Channel API. If one of these APIs is called with a wireless interface that is not the primary interface, the API will fail. To get the primary interface, use the Get Primary Interface API.
It may be necessary to get a list of all wireless interfaces. The Get Interface by Index API facilitates this. It takes as a parameter an index, an unsigned integer. If the index is 0, the primary wireless interface will be returned. For index greater than 0, the API returns the corresponding wireless interface, or `-ERANGE`, numeric value out of range, if the index is too large.

9.8.4 APIs Only Available on the Primary Interface

The following APIs only work when called with the primary interface:

```
Get WiFi Bandwidth
Set WiFi Bandwidth
Get WiFi Channel
Set WiFi Channel
Start Scan
Get Regulatory TX Power
Get Configured TX Power
Get WiFi Current Transmit Power
Set Regulatory Channel
Get Regulatory Region
Set Regulatory Region
Get IEEE 802.11 Standard
Get List Wifi Channels
Get WiFi Noise
Get WiFi RSSI by RF Chain
Get MCS Rate
Set MCS Rate
Reload in WiFi mode
```

9.8.5 Quality of Service (QoS) extensions

A number of APIs handle QoS enhancements parameters for WiFi. The enhancements are based on WMM (Wireless MultiMedia extensions) specification by WiFi aliance, which, in turn, is a subset of 802.11e amendment.
Under WMM, all outgoing traffic is divided into four logical queues, and each queue is set in correspondence with one of four access categories (AC). Specification references access categories by their symbolic names AC_BK, AC_BE, AC_VI and AC_VO, but APIs reference ACs by their corresponding numeric indexes. Mapping between AC's symbolic name and it's index, as well as AC's relative priorities are given in a table:

ACname	ACindex	Priority	Description
AC_BK	1	Lowest	Background traffic
AC_BE	0		Best effort traffic
AC_VI	2		Video traffic
AC_VO	3	Highest	Voice traffic

Access category is merely a label for a certain set of medium access parameters of one respective outgoing traffic queue. Each transmission queue competes for medium access using it's own set of parameters, and variations in ACs parameters value ensures statistical prioritization of one outgoing traffic queue over another.
APIs differentiate between ACs parameters applied to prioritize outgoing traffic by device itself (short, "self params"), and ACs parameters of a BSS the device is associated with (short, "BSS params"). In case of operating in AP mode, device uses "self params" internally for prioritizing it's own outgoing traffic, while it signals "BSS params" to it's client STAs in managment frames headers. In case of operating in STA mode, only "self params" set have a meaning. STA receives it's QoS parameters from associated AP, and it doesn't use its "BSS params" set in any way. Still, APIs allow to set/get both "self" and "BSS" parameters sets in either STA or AP mode.
There are in total six QoS parameters, and APIs use numeric indexes to reference them. Mapping of parameters symbolic names to their corresponding indexes are showed in a table:

Name	Index	Description
ECWMin	1	Contention window exponent (MIN value)
ECWMax	2	Contention window exponent (MAX value)
AIFS	3	Arbitration InterFrame Spacing number
TXOP	4	Transmit Opportunity limit
ACM	5	Admission Control Mandatory
AckPolicy	6	Frame Ack Policy

Following is a short description of AC parameters and their possible values. Parameters are per-VAP, per-AC.

9.8.5.1 ECWMin

Exponent of minimum possible value for contention window (CWMin). This encodes the values of CWMin as an exponent: CWMin = 2^ECWMin - 1.
For example, if ECWMin is 8, then CWMin is 2^8 - 1, or 255. Possible values are 0-15.

9.8.5.2 ECWMax

Exponent of maximum possible value for contention window (CWMax). This encodes the values of CWMax as an exponent: CWMax = 2^ECWMax - 1.
For example, if ECWMax is 8, then CWMax is 2^8 - 1, or 255. Possible values are 0-15.

9.8.5.3 AIFS

Arbitration Inter Frame Spacing Number - the number of time slots in the arbitration interframe space.
Possible values are 0-15.

9.8.5.4 TXOP

Transmit Opportunity limit, in microseconds. A limit to a length of time interval during which device can acquire medium for private use.
Possible values are 0-8192.

9.8.5.5 ACM

Admission Control Mandatory flag. Applies to "BSS params" set only and allows AP to signal to it's client STAs that admission control is mandatory for some AC, and STA has to start traffic transmission for this particular AC by issuing TSPEC request first.
Possible values are 0 and 1. APIs will not allow to set or query ACM parameter for "self params" set.

9.8.5.6 AckPolicy

Acknowledge policy expected by sender device. Only meaningful for outgoing traffic. Whether or not sender expects the receiver to send an ACK in response to normally received frame.
Possible values are 0 and 1. APIs will not allow to set or query AckPolicy parameter for "BSS params" set.

9.8.6 Function Documentation

9.8.6.1 qcsapi_wifi_get_mode()

```
int qcsapi_wifi_get_mode (
            const char * ifname,
            qcsapi_wifi_mode * p_wifi_mode )
```

Determine what mode the WiFi interface is operating in, access point or station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_wifi_mode	return parameter to contain the operational mode of the interface.

Returns

0 if the value in p_wifi_mode is successfully filled in with the correct value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mode <WiFi interface>
```

Output is either `Access Point` or `Station`, unless an error occurs.

9.8.6.2 qcsapi_wifi_set_mode()

```
int qcsapi_wifi_set_mode (
            const char * ifname,
            const qcsapi_wifi_mode new_wifi_mode )
```

Sets the mode for the WiFi interface.

Note

As a side effect of this API call, a new network interface may be created.

The API will fail if the referenced interface already exists.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_wifi_mode	the wifi mode to set the interface to.

Returns

0 if the WiFi interface mode is set successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_mode <WiFi interface> <ap | sta>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.3 qcsapi_wifi_get_phy_mode()

```
int qcsapi_wifi_get_phy_mode (
            const char * ifname,
            char * p_wifi_phy_mode )
```

Determine what mode the WLAN interface is operating in.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_wifi_phy_mode	return parameter to contain the current phy mode of the interface.

Returns

0 if the value in p_wifi_phy_mode is filled the correct value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_phy_mode <WiFi interface>
```

Output is either `11a` or `11na` or `11b` or `11g` or `11ng` or `11ac` or `11acEdge+` or `11acEdge-` or `11acCntr+` or `11acCntr-`, unless an error occurs.

9.8.6.4 qcsapi_wifi_set_phy_mode()

```
int qcsapi_wifi_set_phy_mode (
            const char * ifname,
            const char * new_phy_mode )
```

Sets the phy mode for the WiFi interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_phy_mode	the wifi phy mode to set the interface to.

Returns

0 if the WiFi interface phy mode is set successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_phy_mode <WiFi interface> <11b | 11a | 11g | 11na | 11ng |
11ac | 11acEdge+ | 11acEdge- | 11acCntrl+ | 11acCntrl- >
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.5 qcsapi_wifi_reload_in_mode()

```
int qcsapi_wifi_reload_in_mode (
            const char * ifname,
            const qcsapi_wifi_mode new_wifi_mode )
```

This function will delete the interface and other WDS vaps that may exist, then recreate the interface in the requested mode, and perform all necessary initialization.

This API is used for switching between AP and STA modes without rebooting.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
new_wifi_mode	the new wifi mode to set the interface to.

Returns

0 if the WiFi interface mode is changed successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi reload_in_mode <WiFi interface> <ap | sta>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.6 qcsapi_wifi_rfenable()

```
int qcsapi_wifi_rfenable (
            const qcsapi_unsigned_int onoff )
```

This API call enables or disables the Radio. Disabling the radio kills the daemon process `wpa_supplicant` or `hostapd`, in addition to bring the RF down.

Parameters

onoff	if zero, turn the RF off, else turn the RF on.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Warning

This API relies on the script '/scripts/rfenable' being present on the board to work.

`call_qcsapi` interface:

```
call_qcsapi rfenable <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.7 qcsapi_wifi_rfstatus()

```
int qcsapi_wifi_rfstatus (
            qcsapi_unsigned_int * rfstatus )
```

This API call returns the current status of the device radio.

Parameters

onoff	a pointer to the buffer for storing the returned value

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Warning

This API relies on the script '/scripts/rfstatus' being present on the board to work.

`call_qcsapi` interface:

```
call_qcsapi rfstatus
```

Unless an error occurs, the output will be the current status of radio.

9.8.6.8 qcsapi_wifi_get_bw()

```
int qcsapi_wifi_get_bw (
            const char * ifname,
            qcsapi_unsigned_int * p_bw )
```

Return the configured bandwidth, as specified in the 802.11n standard, either 20 MHz or 40 MHz.

Returned value is in MHz, either 20 40, or 80.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_bw	return parameter to contain the bandwidth the interface is working in (20, 40 or 80)

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_bw <WiFi interface>
```

Unless an error occurs, the output will be the current bandwidth configuration, either 20, 40 or 80.

See also

qcsapi_wifi_set_bw

qcsapi_wifi_set_vht

9.8.6.9 qcsapi_wifi_set_bw()

```
int qcsapi_wifi_set_bw (
            const char * ifname,
            const qcsapi_unsigned_int bw )
```

Use this API to set the bandwidth during system startup, before calling the Enable Interface API for the WiFi device.

Bandwidth can be either 40MHz or 20MHz.

Note

This API can only be used on the primary interface (wifi0)

If the bandwidth is set to 20MHz, any associations will have a bandwidth limited to 20 MHz. If the bandwidth is set to 40MHz, an association will default to a bandwidth of 40 MHz (if the peer supports this). If the bandwidth is set to 80MHz, an association will default to a bandwidth of 80 MHz (if the peer supports this).

Parameters

ifname	the primary WiFi interface, wifi0 only.
bw	the bandwith to set the device to.

Note

80MHz bandwidth is only supported in 802.11ac mode. 802.11ac mode is set using qcsapi_wifi_set_vht

`call_qcsapi` interface:

```
call_qcsapi set_bw <WiFi interface> <80 | 40 | 20>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_wifi_set_vht

9.8.6.10 qcsapi_wifi_get_24g_bw()

```
int qcsapi_wifi_get_24g_bw (
            const char * ifname,
            qcsapi_unsigned_int * p_bw )
```

Return the configured bandwidth, as specified in the 802.11b/g/ng standard, either 20 MHz or 40 MHz.

Returned value is in MHz, either 20 or 40. This API is only available on the primary WiFi interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_bw	return parameter to contain the bandwidth the interface is working in (20 or 40)

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_24g_bw <WiFi interface>
```

Unless an error occurs, the output will be the current bandwidth configuration, either 20 or 40.

9.8.6.11 qcsapi_wifi_set_24g_bw()

```
int qcsapi_wifi_set_24g_bw (
            const char * ifname,
            const qcsapi_unsigned_int bw )
```

Use this API to set the bandwidth during system startup, before calling the Enable Interface API for the WiFi device.

Bandwidth can be either 40MHz or 20MHz. This API is only available on the primary WiFi interface.

If the bandwidth is set to 20MHz, any associations will have a bandwidth limited to 20 MHz. If the bandwidth is set to 40MHz, an association will default to a bandwidth of 40 MHz (if the peer supports this).

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
bw	the bandwith to set the device to.

`call_qcsapi` interface:

```
call_qcsapi set_24g_bw <WiFi interface> <40 | 20>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.12 qcsapi_wifi_set_vht()

```
int qcsapi_wifi_set_vht (
            const char * ifname,
            const qcsapi_unsigned_int the_vht )
```

This API call is used to set vht mode to enable 802.11ac operation.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

Note

The bandwidth to use is set independently, using qcsapi_wifi_set_bw

`call_qcsapi` interface:

```
call_qcsapi set_vht <wifi interface> <0 | 1>
```

See also

qcsapi_wifi_set_bw

9.8.6.13 qcsapi_wifi_get_vht()

```
int qcsapi_wifi_get_vht (
            const char * ifname,
            qcsapi_unsigned_int * vht )
```

This API call is used to get vht mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be `1` (enabled) or `0` (disabled)

`call_qcsapi` interface:

```
call_qcsapi get_vht <wifi interface>
```

9.8.6.14 qcsapi_wifi_get_channel()

```
int qcsapi_wifi_get_channel (
            const char * ifname,
            qcsapi_unsigned_int * p_current_channel )
```

Get the current channel number.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_current_channel	a pointer to the buffer for storing the returned value

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages. If the channel has not been set properly, this API can return -ERANGE, numeric value out of range.

`call_qcsapi` interface:

```
call_qcsapi get_channel <WiFi interface>
```

The output will be the channel number unless an error occurs.

9.8.6.15 qcsapi_wifi_set_channel()

```
int qcsapi_wifi_set_channel (
            const char * ifname,
            const qcsapi_unsigned_int new_channel )
```

Set the current channel.

Note

This API can only be used on the primary interface (wifi0)

The channel can only be set when the interface is up.

Parameters

ifname	the primary WiFi interface, wifi0 only.
new_channel	the new channel to be used

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages. If the channel has not been set properly, this API can return -ERANGE, numeric value out of range.

The new channel must be between 1 and 255, and is further restricted by the 802.11 standard.

This is an engineering API, since it does not account for regulatory requirements. Use of this API can cause the system to violate regulatory requirements. Use of the Set Regulatory Channel API is recommended.

`call_qcsapi` interface:

```
call_qcsapi set_channel <WiFi interface> <new_channel>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.16 qcsapi_wifi_get_chan_pri_inactive()

```
int qcsapi_wifi_get_chan_pri_inactive (
            const char * ifname,
            struct qcsapi_data_256bytes * buffer )
```

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0). only
buffer	buffer to retrieve inactive channel list

Returns

0 if the channel list was retrieved.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_chan_pri_inactive <WiFi interface>
```

Unless an error occurs, the output will be the string of channel list with flag "auto" if it's an auto channel.

9.8.6.17 qcsapi_wifi_set_chan_pri_inactive()

```
int qcsapi_wifi_set_chan_pri_inactive (
            const char * ifname,
            const qcsapi_unsigned_int channel,
            const qcsapi_unsigned_int inactive )
```

Specify whether the channel can be used as primary channel or not.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
channel	the channel to change the inactive flag
inactive	the flag whether it can be used as primary channel or not

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

The channel must be between 1 and 255, and is further restricted by the 802.11 standard.

The inactive flag must be either 0 or 1.

`call_qcsapi` interface:

```
call_qcsapi set_chan_pri_inactive <WiFi interface> <channel> <inactive>
```

If inactive flag is not specified, default is 1.

Unless an error occurs, the output will be the string `complete`.

9.8.6.18 qcsapi_wifi_set_chan_pri_inactive_ext()

```
int qcsapi_wifi_set_chan_pri_inactive_ext (
            const char * ifname,
            const qcsapi_unsigned_int channel,
            const qcsapi_unsigned_int inactive,
            const uint32_t option_flags )
```

Specify whether the channel can be used as primary channel or not, and also pass control flags to kernel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	the channel to change the inactive flag
inactive	the flag whether it can be used as primary channel or not
option_flags	the control flags .

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

This API is only available on the primary WiFi interface.

The channel must be between 1 and 255, and is further restricted by the 802.11 standard.

The inactive flag must be either 0 or 1.

the option_flags is defined as below: 0x1 - auto channel selection only. If it is set, the primary channel inactive flag is valid for auto channel selection only, it is invalid for manual channel configuration. If it is not set, the inactive flag is valid for both auto and manual channel configuration.

`call_qcsapi` interface:

```
call_qcsapi set_chan_pri_inactive <WiFi interface> <channel> <inactive>
<option>
```

If inactive flag is not specified, default is 1. The option can be "autochan", and it is valid only when inactive flag is 1. The inactive flag must be present if option is present.

Unless an error occurs, the output will be the string `complete`.

9.8.6.19 qcsapi_wifi_chan_control()

```
int qcsapi_wifi_chan_control (
            const char * ifname,
            const struct qcsapi_data_256bytes * chans,
            const uint32_t cnt,
            const uint8_t flag )
```

Specify the channel can be used or not, this API can only be used during start-up period.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
chans	point to array of channels.
cnt	the number of channels
flag	indication of enabling or disabling channels, 0: enable, 1: disable

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

The chans must be between -255 and 255, and the absolute value is further restricted by the 802.11 standard.

`call_qcsapi` interface:

```
call_qcsapi set_chan_disabled <WiFi interface> <channel> ...
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.20 qcsapi_wifi_get_chan_disabled()

```
int qcsapi_wifi_get_chan_disabled (
            const char * ifname,
            struct qcsapi_data_256bytes * p_chans,
            uint8_t * p_cnt )
```

List the channels which are disabled and can not be used for primary channel.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_chans	a pointer to the buffer for storing array of channels.
p_cnt	a pointer to the buffer for storing the number of disabled channels

Returns

0 if the channel number was returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_chan_disabled <WiFi interface> <channel> ...
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.21 qcsapi_wifi_get_supported_freq_bands()

```
int qcsapi_wifi_get_supported_freq_bands (
            const char * ifname,
            string_32 p_bands )
```

List the supported frequency bands for a given interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_chans	a pointer to the buffer for storing array of channels.

Returns

0 if supported frequency band string would be returned.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_supported_freq_bands <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.22 qcsapi_wifi_get_beacon_interval()

```
int qcsapi_wifi_get_beacon_interval (
            const char * ifname,
            qcsapi_unsigned_int * p_current_intval )
```

Get the beacon interval in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_current_intval	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_beacon_interval <WiFi interface>
```

The output will be the beacon interval in milliseconds unless an error occurs.

9.8.6.23 qcsapi_wifi_set_beacon_interval()

```
int qcsapi_wifi_set_beacon_interval (
            const char * ifname,
            const qcsapi_unsigned_int new_intval )
```

Set the beacon interval in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_intval	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_beacon_interval <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.24 qcsapi_wifi_get_dtim()

```
int qcsapi_wifi_get_dtim (
            const char * ifname,
            qcsapi_unsigned_int * p_dtim )
```

Get the frequency (in number of beacons) at which the DTIM (Delivery Traffic Information Message) Information Element is added to beacons.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_dtim	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_dtim <WiFi interface>
```

The output will be the DTIM interval in milliseconds unless an error occurs.

9.8.6.25 qcsapi_wifi_set_dtim()

```
int qcsapi_wifi_set_dtim (
            const char * ifname,
            const qcsapi_unsigned_int new_dtim )
```

Set the frequency (in number of beacons) at which the DTIM (Delivery Traffic Information Message) Information Element is added to beacons.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_dtim	the new DTIM value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_dtim <WiFi interface> <DTIM interval>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.26 qcsapi_wifi_get_assoc_limit()

```
int qcsapi_wifi_get_assoc_limit (
            const char * ifname,
            qcsapi_unsigned_int * p_assoc_limit )
```

This API retrieves the current association limit for the primary interface. The association limit is the number of stations that may be simultaneously associated to this AP.

Note

This API can only be used in AP mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_assoc_limit	Address of variable for result retrieval

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_dev_assoc_limit wifi0
```

The output will be the overall device association limit, or an error message in case of failure.

See also

qcsapi_wifi_get_bss_assoc_limit

9.8.6.27 qcsapi_wifi_get_bss_assoc_limit()

```
int qcsapi_wifi_get_bss_assoc_limit (
            qcsapi_unsigned_int group,
            qcsapi_unsigned_int * p_assoc_limit )
```

This API retrieves the current VAP logical group association limit. The association limit is the maximum number of stations that can be associated to this VAP even if the device limit is not reached.

Note

This API can only be used in AP mode.

Parameters

group	group number, between 1 and 31
p_assoc_limit	Address of variable for return result

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_bss_assoc_limit <group>
```

The output will be the VAPs logical group association limit of the interface, or an error message in case of failure.

See also

qcsapi_wifi_get_assoc_limit

9.8.6.28 qcsapi_wifi_set_assoc_limit()

```
int qcsapi_wifi_set_assoc_limit (
            const char * ifname,
            const qcsapi_unsigned_int new_assoc_limit )
```

This API sets the current association limit across all BSSes in the system.

Note

When the limit is changed, all the group assoc parameters configured through call_qcsapi set_bss_assoc_limit and call_qcsapi set_SSID_assoc_reserve commands will also be reset to new dev limit and 0 respectively. All devices will be disassociated and forced to reassociate.

This API can only be used in AP mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
new_assoc_limit	New association limit

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_dev_assoc_limit wifi0 <limit>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_wifi_set_bss_assoc_limit

9.8.6.29 qcsapi_wifi_set_bss_assoc_limit()

```
int qcsapi_wifi_set_bss_assoc_limit (
            const qcsapi_unsigned_int group,
            const qcsapi_unsigned_int limit )
```

This API sets the maximum number of associations allowed across a logical group of interfaces.

Note

When the limit is changed, all devices associated to the group will be disassociated and forced to reassociate.

This API can only be used in AP mode.

Parameters

group	is a group number, between 1 and 31
limit	is the maximum number of devices that are allowed to associate with any WiFi interface in the logical group

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_bss_assoc_limit <group> <limit>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_wifi_set_assoc_limit

9.8.6.30 qcsapi_wifi_set_SSID_group_id()

```
int qcsapi_wifi_set_SSID_group_id (
            const char * ifname,
            const qcsapi_unsigned_int group )
```

This API assigns an SSID to a logical group.

Note

This API can only be used in AP mode.

Configuring this parameter, disassociates all associated devices and will force to reassociate.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
group	group number, between 1 and 31

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_SSID_group_id <WiFi interface> <group>
Unless an error occurs, the output will be the string complete.
```

9.8.6.31 qcsapi_wifi_get_SSID_group_id()

```
int qcsapi_wifi_get_SSID_group_id (
            const char * ifname,
            qcsapi_unsigned_int * p_group )
```

This API returns the SSID's logical group id.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_group	Address of variable for result retrieval

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_SSID_group_id <WiFi interface>

The output will be the VAP's logical group id, or an error message in case
of failure.
```

9.8.6.32 qcsapi_wifi_set_SSID_assoc_reserve()

```
int qcsapi_wifi_set_SSID_assoc_reserve (
            const qcsapi_unsigned_int group,
            const qcsapi_unsigned_int value )
```

This API reserves associations for a group.

Note

This API can only be used in AP mode.

Configuring this parameter, disassociates all associated devices and will force to reassociate.

Parameters

group	group number, between 1 and 31
value	number of reserved associations for a group

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_SSID_assoc_reserve <group> <value>

Unless an error occurs, the output will be the string complete.
```

9.8.6.33 qcsapi_wifi_get_SSID_assoc_reserve()

```
int qcsapi_wifi_get_SSID_assoc_reserve (
            qcsapi_unsigned_int group,
            qcsapi_unsigned_int * p_value )
```

This API gets the number of associations reserved for the group.

Note

This API can only be used in AP mode.

Parameters

group	group number, between 1 and 31
p_value	address of variable for return result

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_SSID_assoc_reserve <group>

The output will be the VAPs logical group reserved association value, or an
error message in case of failure.
```

9.8.6.34 qcsapi_wifi_get_BSSID()

```
int qcsapi_wifi_get_BSSID (
            const char * ifname,
            qcsapi_mac_addr current_BSSID )
```

This API retrieves the current BSSID (basic service set identification)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_BSSID	memory in which to store the current BSSID. On an unassociated station, or if the interface is disabled, this field will be zeroed.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_BSSID <WiFi interface>
```

The output will be a printout of the BSSID MAC address on success, or print an error message on failure.

9.8.6.35 qcsapi_wifi_get_config_BSSID()

```
int qcsapi_wifi_get_config_BSSID (
            const char * ifname,
            qcsapi_mac_addr config_BSSID )
```

Retrieves BSSID stored in configuration file

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
config_BSSID	memory in which to store the configured BSSID. If there is no BSSID configured for the interface memory will be filled with zeros.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_config_BSSID <WiFi interface>
```

The output will be a printout of the BSSID MAC address on success, or print an error message on failure.

9.8.6.36 qcsapi_wifi_ssid_get_bssid()

```
int qcsapi_wifi_ssid_get_bssid (
            const char * ifname,
            const qcsapi_SSID ssid_str,
            qcsapi_mac_addr bssid )
```

Retrieve the BSSID per SSID stored in the configuration file (if it exists)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
BSSID	memory in which to store the retrieved BSSID. If there is no BSSID configured for the SSID, return result will be error 1012

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_ssid_BSSID <WiFi interface> <SSID name>
```

The output will be a printout of the BSSID MAC address on success, or print an error message on failure.

9.8.6.37 qcsapi_wifi_ssid_set_bssid()

```
int qcsapi_wifi_ssid_set_bssid (
            const char * ifname,
            const qcsapi_SSID ssid_str,
            const qcsapi_mac_addr bssid )
```

Add/remove BSSID per SSID stored in configuration file (for a station). This optional configuration will restrict to the specified BSSID address.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
SSID_str	is the SSID to be configured
BSSID	memory in which to store the required BSSID. If the BSSID parameter is filled with all 0xFF (ie MAC address ff:ff:ff:ff:ff:ff), then any existing configured BSSID is removed for the specified SSID.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_ssid_BSSID <WiFi interface> <SSID name> <BSSID MAC address>
```

9.8.6.38 qcsapi_wifi_get_SSID()

```
int qcsapi_wifi_get_SSID (
            const char * ifname,
            qcsapi_SSID SSID_str )
```

Get the current SSID of the interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
SSID_str	return parameter to contain the SSID.

Returns

0 if the command succeeded and SSID_str contains the SSID.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

On a STA with security enabled and is not in an association, this API will return the 0-length string in the SSID_str parameter.

`call_qcsapi` interface:

```
call_qcsapi get_SSID <WiFi interface>
```

Unless an error occurs, the output will be the current SSID. On a STA, if security is enabled and is not in association, the output will be a blank line.

9.8.6.39 qcsapi_wifi_set_SSID()

```
int qcsapi_wifi_set_SSID (
            const char * ifname,
            const qcsapi_SSID SSID_str )
```

Set the current SSID for AP operation on the given interface.

Note

This API can only be used in AP mode.

Warning

Any preexisting associations will be dropped, and those stations will be required to reassociate, using the new SSID.

Use the WiFi Associate API (qcsapi_wifi_associate) on a STA to set the SSID.

The SSID must be a string with between 1 and 32 characters. Control characters (^C, ^M, etc.) are not permitted.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
SSID_str	the new SSID to set on the interface.

Returns

0 if the command succeeded and the SSID is updated.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_SSID <WiFi interface> <SSID name>
```

Unless an error occurs, the output will be the string `complete`.

Examples

To set SSID to Quantenna, enter:

```
call_qcsapi set_ssid wifi0 "Quantenna"
```

To set SSID to 'Quantenna', enter:

```
call_qcsapi set_ssid wifi0 "'Quantenna'"
```

To set SSID to "Quantenna", enter:

```
call_qcsapi set_ssid wifi0 "\"Quantenna""
```

See also

qcsapi_wifi_associate

9.8.6.40 qcsapi_wifi_get_IEEE_802_11_standard()

```
int qcsapi_wifi_get_IEEE_802_11_standard (
            const char * ifname,
            char * IEEE_802_11_standard )
```

Get the current IEEE 802.11 standard(s) that the WiFi device supports. Expected return values (all are character strings) follow:

Value	Interpretation
a-only	Only 802.11a support present (5 GHz with up to 54 Mbps throughput)
b-only	Only 802.11b support present (2.4 GHz with up to 11 Mbps throughput)
g-only	Only 802.11g support present (2.4GHz with up to 54 Mbps throughput)
n-only	Stations / Access Points are required to support 802.11n
a|n	802.11n with 802.11a available for backwards compatibility
g|n	802.11n with 802.11g available for backwards compatibility

Currently "n-only" and "a|n" are the only expected return values.

Note

This API can only be used on the primary interface (wifi0)

The passed in string MUST be at least 7 bytes to contain the maximum length string per the preceding table.

Parameters

ifname	the primary WiFi interface, wifi0 only.
IEEE_802_11_standard	return parameter to contain the string with the PHY mode.

Returns

0 if the command succeeded and the IEEE_802_11_standard string is filled in successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_standard <WiFi interface>

call_qcsapi get_802.11 <WiFi interface>
```

9.8.6.41 qcsapi_wifi_get_list_channels()

```
int qcsapi_wifi_get_list_channels (
            const char * ifname,
            string_1024 list_of_channels )
```

Get the list of available channels with each value separated by commas.

Note

This API can only be used on the primary interface (wifi0)

Warning

This API does not respect regulatory requirements. Use the get list regulatory channels API (qcsapi_wifi_↩ get_list_regulatory_channels) to get the list of valid channels by regulatory authority.

Parameters

ifname	the primary WiFi interface, wifi0 only.
list_of_channels	return parameter to contain the list of channels, comma separated.

Returns

0 if the command succeeded and the list_of_channels is filled in correctly.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_channel_list <WiFi interface>
```

Unless an error occurs, the output will be the list of WiFi channels per the 802.11 standard.

9.8.6.42 qcsapi_wifi_get_supp_chans()

```
int qcsapi_wifi_get_supp_chans (
            const char * ifname,
            qcsapi_mac_addr mac_addr,
            string_1024 list_of_channels )
```

Get the list of supported channels with each value separated by commas.

Note

This API can only be used on the primary interface (wifi0)

This API can only be used in AP mode.

Warning

This API does not respect regulatory requirements. Use `qcsapi_wifi_get_supp_chans` to get the list of supported channels for a STA.

Parameters

ifname	the primary WiFi interface, wifi0 only.
mac_addr	MAC address of an associated station
list_of_channels	buffer to contain the list of comma-separated channels

Returns

0 if the command succeeded

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_supp_chan <WiFi interface> <MAC address>
```

Unless an error occurs, the output will be the list of WiFi channels per the 802.11 standard.

See Format for a MAC address for details of formatting MAC addresses.

9.8.6.43 qcsapi_wifi_get_mode_switch()

```
int qcsapi_wifi_get_mode_switch (
            uint8_t * p_wifi_mode_switch_setting )
```

This API retrieves the current WiFi mode switch setting.

Note

It is required that the relevant GPIOs attached to the switch (12 and 13 for 3way switch and only 12 for 2way switch) be configured as inputs for this API to work.

Parameters

p_wifi_mode_switch_setting	pointer to result memory. On success, the result memory will have 0, 1 or 2 written to it depending on the position of the switch. If the GPIOs are not configured as inputs, value is undefined. 0 value means AP, 1 means STA and 2 means AUTO mode. On 2way switch only AP and STA modes are existing.

Returns

0 if the command succeeded and the GPIOs are set properly.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mode_switch
```

On success, call_qcsapi will print the setting as an integer (1, 2, or 3) depending on the switch setting, or 0 if the relevant GPIOs are not configured as inputs. On failure, an error message will be written to stdout.

9.8.6.44 qcsapi_wifi_disassociate()

```
int qcsapi_wifi_disassociate (
            const char * ifname )
```

This API call ends the current association and puts the device into an idle state, so that it does not send out probe requests or attempt to associate with any AP.

Note

This API only applies for a STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi disassociate <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.45 qcsapi_wifi_disassociate_sta()

```
int qcsapi_wifi_disassociate_sta (
            const char * ifname,
            qcsapi_mac_addr mac )
```

This API call ends association with remote station. It is a wrapper around hostapd's "disassociate" function.

Note

This API only applies for an AP.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
MAC	\02:51:55:41:00:4C

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi disassociate_sta <WiFi interface> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

Example MAC address: `02:51:55:41:00:4C`. See Format for a MAC address for details of formatting MAC addresses.

9.8.6.46 qcsapi_wifi_reassociate()

```
int qcsapi_wifi_reassociate (
            const char * ifname )
```

On AP reassociation of all clients is forced by disassociating them without deauthenticating. For STA it forces reassociation without going through scan. It is equivalent to calling command `iwpriv <WiFi interface> cl_remove 1`.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi reassociate <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.47 qcsapi_wifi_get_disconn_info()

```
int qcsapi_wifi_get_disconn_info (
            const char * ifname,
            qcsapi_disconn_info * disconn_info )
```

This API is called to get disconnection count. Each time successfully invoke this API, the sequence of disconn_info will increase 1, by which the caller can determine whether module has reset. Note that disconnection count only increases when the peer is authorized and disassocated later.

In STA mode, the disconn_count of disconn_info indicates how many times STA is disconnected from AP since module boot up, the asso_sta_count is 1 if it connects to AP or 0 if not. In AP mode, the disconn_count records times that AP disconnects STA which has been associated with it, the asso_sta_count means the number of STAs connect to AP.

This API also can reset sequence, and asso_sta_count by setting member resetflag of disconn_info with 1.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
disconn_info	where to save the data.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_disconn_info <WiFi interface>
call_qcsapi reset_disconn_info <WiFi interface>
```

9.8.6.48 qcsapi_wifi_disable_wps()

```
int qcsapi_wifi_disable_wps (
            const char * ifname,
            int disable_wps )
```

Available on APs only. This API alters hostapd wps_state.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
disable_wps	0 = set wps_state to configured. 1 = set wps_state to disabled.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi disable_wps <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.49 qcsapi_wifi_associate()

```
int qcsapi_wifi_associate (
            const char * ifname,
            const qcsapi_SSID join_ssid )
```

This API call causes the STA to attempt to associate with the selected SSID, as identified by the input parameter.

The SSID must be present (and configured correctly) on the STA.

Note

This API only applies for a STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
join_ssid	the SSID for the STA to join.

Returns

0 if the command succeeded (association is started - not that the association succeeds).

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi associate <WiFi interface> <SSID name>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.50 qcsapi_wifi_start_cca()

```
int qcsapi_wifi_start_cca (
            const char * ifname,
            int channel,
            int duration )
```

This API causes a CCA measurement to be scheduled 1 second after invocation, on the requested channel, for a requested number of milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	Channel to switch to during CCA measurement
duration	Time in milliseconds to spend on the channel

Returns

0 if the command succeeded (CCA measurement is triggered)

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi start_cca <WiFi interface> channel duration
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.51 qcsapi_wifi_get_noise()

```
int qcsapi_wifi_get_noise (
            const char * ifname,
            int * p_noise )
```

This API reports the noise on the current channel in dBm.

Since the noise is expected to be much less than 1 milliwatt (0 dBm), the value should be much less than 0.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_noise	return parameter to contain the noise figure read from the interface.

Returns

0 if the command succeeded and p_noise contains a valid noise figure.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_noise <WiFi interface>
```

Unless an error occurs, the output will be the current noise in dBm, most likely a negative number.

9.8.6.52 qcsapi_wifi_get_rssi_by_chain()

```
int qcsapi_wifi_get_rssi_by_chain (
            const char * ifname,
            int rf_chain,
            int * p_rssi )
```

This API reports the RSSI of the selected RF chain in dBm.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
rf_chain	the RF chain to get the RSSI of (between 0 and 3).
p_rssi	return parameter to contain the RSSI reading of the interface/RF chain pair.

Returns

0 if the command succeeded and p_rssi contains a valid value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rssi_by_chain <WiFi interface> <RF Chain number>
```

Unless an error occurs, the output will be the RSSI of the selected RF chain in dBm.

9.8.6.53 qcsapi_wifi_get_avg_snr()

```
int qcsapi_wifi_get_avg_snr (
            const char * ifname,
            int * p_snr )
```

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_snr	return parameter to contain the average SNR of the interface.

Returns

0 if the command succeeded and p_snr contains a valid value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_avg_snr <WiFi interface>
```

Unless an error occurs, the output will be the average SNR of the primary WiFi interface.

9.8.6.54 qcsapi_get_primary_interface()

```
int qcsapi_get_primary_interface (
            char * ifname,
            size_t maxlen )
```

This API will return the name of the primary WiFi interface.

The primary interface is usually the first AP-mode interface created on system start up, and is the only WiFi interface that allows configuration of the underlying properties of the radio: channel, TX power, bandwidth, etc.

The primary interface is thus distinct from any additional AP-mode virtual interfaces created as part of the MBSSID feature, which do not allow these radio properties to be set.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
maxlen	the size of the buffer pointed to by ifname.

Returns

0 if the command succeeded and ifname contains the primary WiFi interface.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_primary_interface
```

Output should be the name of the primary interface unless an error occurs.

9.8.6.55 qcsapi_get_interface_by_index()

```
int qcsapi_get_interface_by_index (
            unsigned int if_index,
            char * ifname,
            size_t maxlen )
```

This API will return the name of the WiFi interface that corresponds to the input parameter.

For if_index = 0, this API will return the name of the primary interface.

For if_index $>$ 0, the API will return the corresponding secondary interface.

If if_index exceeds the number of interfaces - 1, this API will return a range error indicating the index parameter is too large.

No holes in the list of entries will be present; starting at 0, for each consecutive value of index, either a unique interface name is returned or the API returns range error.

If for a particular value of index the API returns range error, the API will return the same range error for all larger values of index.

Parameters

if_index	the interface index to get the name of.
ifname	the interface to perform the action on. (e.g. wifi0).
maxlen	the size of the buffer pointed to by ifname.

Returns

0 if the command succeeded and ifname contains the string of the interface correspoinding to index 'if_index'.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_interface_by_index <index>
```

Output should be the name of the interface that corresponds to the index value.

9.8.6.56 qcsapi_wifi_set_wifi_macaddr()

```
int qcsapi_wifi_set_wifi_macaddr (
            const qcsapi_mac_addr new_mac_addr )
```

This API allows the primary WiFi interface MAC address to be set.

Parameters

new_mac_addr	the new MAC address for the primary WiFi interface.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Warning

This API can ONLY be called after the QDRV is started, but before the WiFi mode is selected.

This API cannot be used to change the WiFi MAC address dynamically - it can be called only once, after the QDRV is started. To change the WiFi MAC address again, a reboot is required.

This API does NOT save the set WiFi MAC address across reboots. Additional logic is required to save/restore the MAC address across reboots.

`call_qcsapi` interface:

```
call_qcsapi set_wifi_macaddr <new MAC addr>
```

Output should be the string `complete`.

9.8.6.57 qcsapi_interface_get_BSSID()

```
int qcsapi_interface_get_BSSID (
            const char * ifname,
            qcsapi_mac_addr current_BSSID )
```

Return the Basic Service Set ID (BSSID).

For an AP, this is the MAC address of the WiFi interface. For a STA, it is the MAC address of the AP it is associated with. If the STA is not in association, the BSSID will be the address `00:00:00:00:00:00`

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_BSSID	return parameter to contain the BSSID.

Returns

0 if the command succeeded and current_BSSID contains a valid value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_BSSID <WiFi interface>
```

Unless an error occurs, the output will be the current BSSID, expressed in the standard MAC address format.

On a station not in association, the value will be `00:00:00:00:00:00`

9.8.6.58 qcsapi_wifi_get_rates()

```
int qcsapi_wifi_get_rates (
            const char * ifname,
            qcsapi_rate_type rate_type,
            string_2048 supported_rates )
```

This API will list the supported rates (as a string), with each value in megabits per second, separated by commas.

These rates represent what the device is capable of; the return value is NOT affected by the current rate setting.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
rate_type	set this to qcsapi_possible_rates.
supported_rates	return parameter to contain the comma separated rate list.

Returns

0 if the command succeeded and current_BSSID contains a valid value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rates <WiFi interface> possible_rates
```

Unless an error occurs, the output will be the set of possible rates supported by the interface.

9.8.6.59 qcsapi_wifi_set_rates()

```
int qcsapi_wifi_set_rates (
            const char * ifname,
            qcsapi_rate_type rate_type,
            const string_256 current_rates,
            int num_rates )
```

9.8.6.60 qcsapi_get_max_bitrate()

```
int qcsapi_get_max_bitrate (
            const char * ifname,
            char * max_bitrate,
            const int max_str_len )
```

This API will only output "auto" currently.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_bitrate	String value of the max_bitrate, currently only "auto" can be retrieved.
max_str_len	The length of the string point to max_bitrate.

Returns

0 if the command succeeded and max_bitrate contains a valid value.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_max_bitrate <WiFi interface>
```

Unless an error occurs, the output will be the max_bitrate supported by the interface, currently only outputting "auto".

9.8.6.61 qcsapi_set_max_bitrate()

```
int qcsapi_set_max_bitrate (
            const char * ifname,
            const char * max_bitrate )
```

This API can only input "auto" currently.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_bitrate	String value of the max_bitrate, currently only "auto" can be set.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_max_bitrate <WiFi interface> <max_bitrate>
```

Unless an error occurs, output should be the string `complete`.

9.8.6.62 qcsapi_wifi_qos_get_param()

```
int qcsapi_wifi_qos_get_param (
            const char * ifname,
            int the_queue,
            int the_param,
            int ap_bss_flag,
            int * p_value )
```

Returns the value of a Quality of Service Parameter, based on the selected queue.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_queue	which queue to use. Value ranges from 0 to 3, see section Quality of Service (QoS) extensions for queue index to symbolic name mappings.
the_param	which parameter to report. Value ranges from 1 to 6, refer to sections ECWMin through AckPolicy for description and limitations.
ap_bss_flag	set to "0" or "1" to report either egress (self) or ingress (BSS) QoS parameter respectively. Refer to section Quality of Service (QoS) extensions for difference.
p_value	address of the location to receive the value of the QOS parameter.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_qos_param <WiFi interface> <0|1|2|3> <1|2|3|4|6> [<0|1>]
```

9.8.6.63 qcsapi_wifi_qos_set_param()

```
int qcsapi_wifi_qos_set_param (
            const char * ifname,
            int the_queue,
            int the_param,
            int ap_bss_flag,
            int value )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_queue	which queue to use. Value ranges from 0 to 3, see section Quality of Service (QoS) extensions for queue index to symbolic name mappings.
the_param	which parameter to report. Value ranges from 1 to 6, refer to sections ECWMin through AckPolicy for description and limitations.
ap_bss_flag	"0" or "1" to set either egress (self) or ingress (BSS) QoS parameter respectively. Refer to section Quality of Service (QoS) extensions for difference.
value	the value of the QOS parameter to set.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_qos_param <WiFi interface> <0|1|2|3> <1|2|3|4|6> <value> [<0|1>]
```

9.8.6.64 qcsapi_wifi_get_wmm_ac_map()

```
int qcsapi_wifi_get_wmm_ac_map (
            const char * ifname,
            string_64 mapping_table )
```

Returns the current mapping table from TOS/DSCP or IEEE802.1p priority to WMM AC index

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mapping_table	Return value to contain the mapping table for priorities. Must be a memory area large enough to contain 64 byte values

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_wmm_ac_map <WiFi interface>
```

9.8.6.65 qcsapi_wifi_set_wmm_ac_map()

```
int qcsapi_wifi_set_wmm_ac_map (
            const char * ifname,
            int user_prio,
            int ac_index )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
user_prio	Which user priority to be set. Value ranges from 0 to 7.
ac_index	The AC index to map to the input user_prio. Valid values are 0 to 3, see Quality of Service (QoS) extensions for correspondence between AC index and AC symbolic name.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_wmm_ac_map <WiFi interface> <0|1|2|3|4|5|6|7> <0|1|2|3>
```

9.8.6.66 qcsapi_wifi_get_dscp_8021p_map()

```
int qcsapi_wifi_get_dscp_8021p_map (
            const char * ifname,
            string_64 mapping_table )
```

Returns the current mapping table from IP DSCP to IEEE802.1p user priority

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mapping_table	Return value to contain the mapping table for IEEE802.1p user priorities. Must be a memory area large enough to contain 64 byte values

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_dscp_8021p_map <WiFi interface>
```

9.8.6.67 qcsapi_wifi_get_dscp_ac_map()

```
int qcsapi_wifi_get_dscp_ac_map (
            const char * ifname,
            struct qcsapi_data_64bytes * mapping_table )
```

Returns the current mapping table from IP DSCP to WME AC user priority

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mapping_table	Return value to contain the mapping table for WME AC user priorities. Must be a memory area large enough to contain 64 byte values.

Return parameter to contain the mapping table for WME AC user priorities

`call_qcsapi` interface:

```
call_qcsapi get_dscp_ac_map <WiFi interface>
```

9.8.6.68 qcsapi_wifi_set_dscp_8021p_map()

```
int qcsapi_wifi_set_dscp_8021p_map (
            const char * ifname,
            const char * ip_dscp_list,
            uint8_t dot1p_up )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ip_dscp_list	List of IP DSCP values to be set. Value ranges from 0 to 64 and 64 is a special use to revert mapping table to default. If the IP DSCP needed to be set is 40 and 46 the format should be 40,46.
dot1p_up	the 802.1p UP to be mapped to the input IP DSCP. Value ranges from 0∼7.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_dscp_8021p_map <WiFi interface> <0-64>[,1-64]...  <0-7>
```

9.8.6.69 qcsapi_wifi_set_dscp_ac_map()

```
int qcsapi_wifi_set_dscp_ac_map (
            const char * ifname,
            const struct qcsapi_data_64bytes * dscp_list,
            uint8_t dscp_list_len,
            uint8_t ac )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dscp_list	List of IP DSCP value(s) to be set. Value ranges from 0 to 63. If more than one IP DSCP mapping is specified, the values must be separated by commas.
dcsp_list_len	The number of IP DSCP values in the dscp_list argument.
ac	the WME AC value that will be mapped to the input IP DSCP. Value ranges from 0 to 3, see Quality of Service (QoS) extensions for correspondence between AC index and AC symbolic name.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

Any DSCP mappings not explicitly set will map to AC_BE.

This API is superseded by qcsapi_wifi_set_qos_map

`call_qcsapi` interface:

```
call_qcsapi set_dscp_ac_map <WiFi interface> <0-64>[,0-64]...  <0-3>
```

As an example, to map the following:

CS0,CS1,CS4,CS7 (map to AC_BE);

AF11,CS2, AF21,CS3,CS6 (map to AC_VI); and

CS5, EF (map to AC_VO)

the following set of call_qcsapi calls are made:

```
call_qcsapi set_dscp_ac_map wifi0 0,8,32,56 0
call_qcsapi set_dscp_ac_map wifi0 10,16,18,24,48 2
call_qcsapi set_dscp_ac_map wifi0 40,46 3
```

9.8.6.70 qcsapi_wifi_set_qos_map()

```
int qcsapi_wifi_set_qos_map (
            const char * ifname,
            const char * qos_map_str )
```

Configure QoS Map Set information element. Refer to IEEE802.11-2012, 8.4.2.97 for further description. This also configures DSCP to TID mapping to be used by AP.

QoS Map Set string have the following format:

```
[<DSCP Exceptions[DSCP,UP]>,] <UP 0 range[low,high]>,...<UP 7 range[low,high]>
```

There can be up to 21 optional DSCP Exceptions which are pairs of DSCP Value (0..63 or 255) and User Priority (0..7). This is followed by eight DSCP Range descriptions with DSCP Low Value and DSCP High Value pairs (0..63 or 255) for each UP starting from 0. If both low and high value are set to 255, the corresponding UP is not used.

Parameters

ifname	the primary WiFi interface, wifi0 only.
qos_map_str	QoS Map Set string

Note

This API supersedes qcsapi_wifi_set_dscp_ac_map that configures the same DSCP to TID mapping. If TID cannot be used it will be replaced by the one mapped to the same AC.

This API can only be used in AP mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_qos_map <WiFi interface> <qos_map_set>
```

Example:

```
call_qcsapi set_qos_map wifi0 53,2,22,6,8,15,0,7,255,255,16,31,32,39,255,255,40,47,255,255
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.71 qcsapi_wifi_del_qos_map()

```
int qcsapi_wifi_del_qos_map (
            const char * ifname )
```

Remove QoS Map Set parameter.

Parameters

ifname	the primary WiFi interface, wifi0 only.

Note

This API can only be used in AP mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi del_qos_map <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.72 qcsapi_wifi_get_qos_map()

```
int qcsapi_wifi_get_qos_map (
            const char * ifname,
            string_256 value )
```

Get the value of QoS Map Set. Format is described in qcsapi_wifi_set_qos_map.

Parameters

ifname	the primary WiFi interface, wifi0 only.
value	pointer to buffer to store returned value

Note

This API can only be used in AP mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_qos_map <WiFi interface>
```

Unless an error occurs, the output will be the value of QoS Map Set.

9.8.6.73 qcsapi_wifi_send_qos_map_conf()

```
int qcsapi_wifi_send_qos_map_conf (
            const char * ifname,
            const qcsapi_mac_addr sta_mac_addr )
```

Send QoS Map Configure action frame to an associated station specified by MAC address.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
sta_mac_addr	STA MAC address

Note

This API can only be used in AP mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi send_qos_map_conf <WiFi interface> <STA MAC addr>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.74 qcsapi_wifi_get_dscp_tid_map()

```
int qcsapi_wifi_get_dscp_tid_map (
            const char * ifname,
            struct qcsapi_data_64bytes * dscp2tid_ptr )
```

Get the value of DSCP to TID mapping. The returned buffer will contain a 64-byte array of 8-bit unsigned integers, which are the TID values for each DSCP from 0 to 63.

Parameters

ifname	the primary WiFi interface, wifi0 only.
dscp2tid_ptr	pointer to buffer to store returned value

Note

This API can only be used in AP mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_dscp_tid_map <WiFi interface>
```

Unless an error occurs the output will be the header: DSCP: TID followed by corresponding <DSCP>: <TID> values on separate lines.

9.8.6.75 qcsapi_wifi_get_priority()

```
int qcsapi_wifi_get_priority (
            const char * ifname,
            uint8_t * p_priority )
```

Get the priority for the given WiFi interface. The priority is used to differentiate traffic between different SSIDs. Traffic in SSIDs with higher priority takes precedence over traffic in SSIDs with lower priority.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_priority	a pointer to the buffer for storing the returned value.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_priority <WiFi interface>
```

The output will be the priority unless an error occurs.

9.8.6.76 qcsapi_wifi_set_priority()

```
int qcsapi_wifi_set_priority (
            const char * ifname,
            uint8_t priority )
```

Set the priority for the given WiFi interface. The priority is used to differentiate traffic between different SSIDs. Traffic in SSIDs with higher priority takes precedence over traffic in SSIDs with lower priority.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
priority	interface priority. Value ranges from 0 to 3.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_priority <WiFi interface> <priority>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.77 qcsapi_wifi_get_airfair()

```
int qcsapi_wifi_get_airfair (
            const char * ifname,
            uint8_t * p_airfair )
```

Get the airtime fairness status for the given WiFi interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_airfair	a pointer to the buffer for storing the returned value.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_airfair <WiFi interface>
```

The output will be the status of airtime fairness unless an error occurs. 0 means diabled, and 1 means enabled.

9.8.6.78 qcsapi_wifi_set_airfair()

```
int qcsapi_wifi_set_airfair (
            const char * ifname,
            uint8_t airfair )
```

Set the airtime fairness for the given WiFi interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
airfair	interface airfair status. Value is either 0(disable) or 1(enable).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_airfair <WiFi interface> <airfair>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.79 qcsapi_wifi_get_tx_power()

```
int qcsapi_wifi_get_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            int * p_tx_power )
```

This API returns the transmit power for the specified channel, which is the maximum allowed power used in the case of beamforming off and 1 spatial stream for current bandwidth. The TX powers are reported in dBm.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is returned.
p_tx_power	return parameter to contain the transmit power.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_tx_power <WiFi interface> <channel>
```

Unless an error occurs, the output will be the transmit power.

Note

This API is deprecated and replaced with qcsapi_wifi_get_tx_power_ext.

9.8.6.80 qcsapi_wifi_set_tx_power()

```
int qcsapi_wifi_set_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const int tx_power )
```

This API call sets the transmit power for a particular channel. The TX power is in dBm unit.

Note

This API can only be used on the primary interface (wifi0)

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is set.
tx_power	tx power to be set.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_tx_power <WiFi interface> <channel> <tx_power>
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_tx_power_ext.

9.8.6.81 qcsapi_wifi_get_bw_power()

```
int qcsapi_wifi_get_bw_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            int * p_power_20M,
            int * p_power_40M,
            int * p_power_80M )
```

This API returns the transmit powers for the specified channel, which is the maximum allowed powers used in the case of beamforming off and 1 spatial stream. The TX powers are reported in dBm.

Note

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is returned.
p_power_20M	return parameter to contain the transmit power for 20MHz bandwidth.
p_power_40M	return parameter to contain the transmit power for 40MHz bandwidth.
p_power_80M	return parameter to contain the transmit power for 80MHz bandwidth.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_bw_power <WiFi interface> <channel>
```

Unless an error occurs, the output will be the transmit powers.

Note

This API is deprecated and replaced with qcsapi_wifi_get_tx_power_ext.

9.8.6.82 qcsapi_wifi_set_bw_power()

```
int qcsapi_wifi_set_bw_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const int power_20M,
            const int power_40M,
            const int power_80M )
```

This API call sets the transmit powers for a particular channel. The TX powers are in dBm unit.

Note

This API can only be used on the primary interface (wifi0)

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is set.
power_20M	tx power for 20MHz bandwidth to be set.
power_40M	tx power for 40MHz bandwidth to be set.
power_80M	tx power for 80MHz bandwidth to be set.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_bw_power <WiFi interface> <channel> <power_20M> <power_↩
40M> <power_80M>
```

Unless an error occurs, the output will be the string `complete`'.

Note

This API is deprecated and replaced with qcsapi_wifi_set_tx_power_ext.

9.8.6.83 qcsapi_wifi_get_bf_power()

```
int qcsapi_wifi_get_bf_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const qcsapi_unsigned_int number_ss,
            int * p_power_20M,
            int * p_power_40M,
            int * p_power_80M )
```

This API returns the transmit powers for the specified channel, which is the maximum allowed powers used in the case of beamforming is on. The TX powers are reported in dBm.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is returned.
number_ss	the number of spatial streams.
p_power_20M	return parameter to contain the transmit power for 20MHz bandwidth.
p_power_40M	return parameter to contain the transmit power for 40MHz bandwidth.
p_power_80M	return parameter to contain the transmit power for 80MHz bandwidth.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_bf_power <WiFi interface> <channel> <number_ss>
```

Unless an error occurs, the output will be the transmit powers.

Note

This API is deprecated and replaced with qcsapi_wifi_get_tx_power_ext.

9.8.6.84 qcsapi_wifi_set_bf_power()

```
int qcsapi_wifi_set_bf_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const qcsapi_unsigned_int number_ss,
            const int power_20M,
            const int power_40M,
            const int power_80M )
```

This API call sets the transmit powers for a particular channel. The TX powers are in dBm unit.

Note

This API can only be used on the primary interface (wifi0)

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is set.
number_ss	the number of spatial streams.
power_20M	tx power for 20MHz bandwidth to be set.
power_40M	tx power for 40MHz bandwidth to be set.
power_80M	tx power for 80MHz bandwidth to be set.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_bf_power <WiFi interface> <channel> <number_ss> <power_↩
20M> <power_40M> <power_80M>
```

Unless an error occurs, the output will be the string `complete`'.

Note

This API is deprecated and replaced with qcsapi_wifi_set_tx_power_ext.

9.8.6.85 qcsapi_wifi_get_tx_power_ext()

```
int qcsapi_wifi_get_tx_power_ext (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const qcsapi_unsigned_int bf_on,
            const qcsapi_unsigned_int number_ss,
            int * p_power_20M,
            int * p_power_40M,
            int * p_power_80M )
```

This API returns the transmit powers for a specified channel, which is the maximum allowed powers used in the specified case. The TX powers are reported in dBm.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is returned.
bf_on	beamforming is either on or off. 1 for beamforming on and 0 for beamforming off.
number_ss	the number of spatial streams.
p_power_20M	return parameter to contains the transmit power for 20MHz bandwidth.
p_power_40M	return parameter to contains the transmit power for 40MHz bandwidth.
p_power_80M	return parameter to contains the transmit power for 80MHz bandwidth.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_tx_power_ext <WiFi interface> <channel> <bf_on(1/0)> <number↩
_ss>
```

Unless an error occurs, the output will be the transmit powers.

9.8.6.86 qcsapi_wifi_set_tx_power_ext()

```
int qcsapi_wifi_set_tx_power_ext (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const qcsapi_unsigned_int bf_on,
            const qcsapi_unsigned_int number_ss,
            const int power_20M,
            const int power_40M,
            const int power_80M )
```

This API call sets the transmit powers for a particular channel. The TX powers are in dBm unit.

Note

This API can only be used on the primary interface (wifi0)

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the tx power is set.
bf_on	beamforming is either on or off. 1 for beamforming on and 0 for beamforming off.
number_ss	the number of spatial streams.
power_20M	tx power for 20MHz bandwidth to be set.
power_40M	tx power for 40MHz bandwidth to be set.
power_80M	tx power for 80MHz bandwidth to be set.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_tx_power_ext <WiFi interface> <channel> <bf_on(1/0)> <number↩
_ss> <power_20M> <power_40M> <power_80M>
```

Unless an error occurs, the output will be the string `complete`'.

9.8.6.87 qcsapi_wifi_get_chan_power_table()

```
int qcsapi_wifi_get_chan_power_table (
            const char * ifname,
            qcsapi_channel_power_table * chan_power_table )
```

This API returns all the transmit powers for a specified channel. The TX powers are reported in dBm.

See the documentation for qcsapi_channel_power_table for full details of the return structure.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
chan_power_table	the pointer to a data structure which is used to store the return powers. the variable "channel" of this structure must be initiated before calling this API.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_chan_power_table <WiFi interface> <channel>
```

Unless an error occurs, the output will be the power table of this channel.

See also

qcsapi_channel_power_table

9.8.6.88 qcsapi_wifi_set_chan_power_table()

```
int qcsapi_wifi_set_chan_power_table (
            const char * ifname,
            qcsapi_channel_power_table * chan_power_table )
```

This API sets all the transmit powers for a particular channel. The TX powers are in dBm unit.

See the documentation for qcsapi_channel_power_table for full details of how to fill out the channel power table.

Note

This API can only be used on the primary interface (wifi0)

The value(s) set by this API are not persistent and must be reapplied if the device reboots.

Parameters

ifname	the primary WiFi interface, wifi0 only.
chan_power_table	the pointer to a data structure which has the channel number and powers.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_chan_power_table <WiFi interface> <channel> <max_power>
<backoff_20M> <backoff_40M> <backoff_80M>
```

`max_power` is the maximum power of this channel's 24 powers.

`backoff_20M` is a 32 bits unsigned value, and every 4 bits indicate the backoff from the `max_power` for a BF/SS case.

The least significant 4 bits are for BF off 1SS case, and the most significant 4 bits are for BF on 4SS case.

For the sequence, please see the enumeration definition for qcsapi_power_indices (qcsapi_power_indices). For example, max_power 23 and backoff_20M 0x54324321 give the powers as below:

the power for 20Mhz bfoff 1ss: 23 - 1 = 22dBm

the power for 20Mhz bfoff 2ss: 23 - 2 = 21dBm

the power for 20Mhz bfoff 3ss: 23 - 3 = 20dBm

the power for 20Mhz bfoff 4ss: 23 - 4 = 19dBm

the power for 20Mhz bfon 1ss: 23 - 2 = 21dBm

the power for 20Mhz bfon 2ss: 23 - 3 = 20dBm

the power for 20Mhz bfon 3ss: 23 - 4 = 19dBm

the power for 20Mhz bfon 4ss: 23 - 5 = 18dBm

`backoff_40M` and `backoff_80M` use the same format as `backoff_20M`, and they are the backoff for 40Mhz and 80Mhz respectively.

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_channel_power_table

9.8.6.89 qcsapi_wifi_get_power_selection()

```
int qcsapi_wifi_get_power_selection (
            qcsapi_unsigned_int * p_power_selection )
```

Get the current mode for selecting power table.

Parameters

p_power_selection	a pointer to the buffer for storing the returned value

Returns

0 if the mode for selecting power table was returned.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi get_power_selection
```

The output will be the mode for selecting power table number unless an error occurs.

9.8.6.90 qcsapi_wifi_set_power_selection()

```
int qcsapi_wifi_set_power_selection (
            const qcsapi_unsigned_int power_selection )
```

Set the mode for selecting power table.

Parameters

power_selection	the mode to be used

Returns

0 if the mode is set successfully.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi set_power_selection <power_selection>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.91 qcsapi_wifi_get_carrier_interference()

```
int qcsapi_wifi_get_carrier_interference (
            const char * ifname,
            int * ci )
```

This API is used to get Carrier/Interference (db).

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ci	return the Carrier/Interference in db unit.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_carrier_db <WiFi interface>
```

Unless an error occurs, the output will be the value of the Carrier/Interference in db unit.

9.8.6.92 qcsapi_wifi_get_congestion_index()

```
int qcsapi_wifi_get_congestion_index (
            const char * ifname,
            int * ci )
```

This API is used to get current congestion index.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ci	return value to contain the congestion index. The congestion index is in the range 0 - 10.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_congest_idx <WiFi interface>
```

Unless an error occurs, the output will be the value of the congestion index.

9.8.6.93 qcsapi_wifi_get_supported_tx_power_levels()

```
int qcsapi_wifi_get_supported_tx_power_levels (
            const char * ifname,
            string_128 available_percentages )
```

This API reports the supported transmit power levels on the current channel as a list of percentages of the maximum allowed by the regulatory authority. A regulatory region must have been configured with the Set Regulatory Region API.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
available_percentages	address of a string to recve the list of available percentages.

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_supported_tx_power <WiFi interface>
```

Unless an error occurs, the output will be the list of percentages of the maximum allowed by the regulatory authority.

9.8.6.94 qcsapi_wifi_get_current_tx_power_level()

```
int qcsapi_wifi_get_current_tx_power_level (
            const char * ifname,
            uint32_t * p_current_percentage )
```

This API reports the transmit power on the current channel as a percentage of the maximum allowed by the regulatory authority. A regulatory region must have been configured with the Set Regulatory Region API.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_current_percentage	return parameter to contains the percentage

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_current_tx_power <WiFi interface>
```

Unless an error occurs, the output will be the percentage of the maximum allowed by the regulatory authority.

9.8.6.95 qcsapi_wifi_set_current_tx_power_level()

```
int qcsapi_wifi_set_current_tx_power_level (
            const char * ifname,
            uint32_t txpower_percentage )
```

This API sets the transmit power on the current channel as a percentage of the maximum allowed by the regulatory authority. A regulatory region must have been configured with the Set Regulatory Region API.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
txpower_percentage	percentage of tx power to set

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_current_tx_power <WiFi interface> <txpower_percentage>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.96 qcsapi_wifi_set_power_constraint()

```
int qcsapi_wifi_set_power_constraint (
            const char * ifname,
            uint32_t pwr_constraint )
```

This API sets the power constraint on the current channel. Power constraint will be filled in power constraint element of beacon and probe response when spectrum management enabled.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
pwr_constraint	power constraint to set

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_power_constraint <WiFi interface> <power constraint>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.97 qcsapi_wifi_get_power_constraint()

```
int qcsapi_wifi_get_power_constraint (
            const char * ifname,
            uint32_t * p_pwr_constraint )
```

This API returns the power constraint on the current channel.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_pwr_constraint	return parameter to contains power constraint

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_power_constraint <WiFi interface>
```

Unless an error occurs, the output will be the power constraint on the current channel.

9.8.6.98 qcsapi_wifi_set_tpc_interval()

```
int qcsapi_wifi_set_tpc_interval (
            const char * ifname,
            int32_t tpc_interval )
```

This API sets the tpc request interval if 802_11h is enabled and periodical method is used.

Note

This API can only be used on the primary interface (wifi0)

This API is available on AP/STA/WDS mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
tpc_interval	interval for tpc request to set

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_tpc_interval <WiFi interface> <tpc_interval>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.99 qcsapi_wifi_get_tpc_interval()

```
int qcsapi_wifi_get_tpc_interval (
            const char * ifname,
            uint32_t * p_tpc_interval )
```

This API gets the tpc request interval if 802_11h is enabled and periodical method is used.

Note

This API can only be used on the primary interface (wifi0)

This API is available on AP/STA/WDS mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
tpc_interval	interval for tpc request to set

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_tpc_interval <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.100 qcsapi_wifi_get_assoc_records()

```
int qcsapi_wifi_get_assoc_records (
            const char * ifname,
            int reset,
            qcsapi_assoc_records * records )
```

This API reports all remote STAs that have associated with the local AP. The MAC address of the WiFi interface is reported together with the last time that STA associated. The STA must pass the 4-way handshake to be included; that is, its security credentials must be correct. Only one entry will be present for a particular STA, based on its MAC address.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
reset	set this to 1 to clear the association records. The current set of association records will be returned.
records	address of the data structure to receive the association records.

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_assoc_records <WiFi interface>
```

Unless an error occur, the output will be a list of all remote STAs that have associated with the device, together with the time they last associated.

9.8.6.101 qcsapi_wifi_get_ap_isolate()

```
int qcsapi_wifi_get_ap_isolate (
            const char * ifname,
            int * p_ap_isolate )
```

Get the current global AP isolation setting

Note

This API can only be used in AP mode.

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
p_ap_isolate	return parameter to contain the current global AP isolation setting

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_ap_isolate <WiFi interface>
```

Unless an error occurs, the output will be 0, 1, representing the qcsapi_ap_isolate_type values.

9.8.6.102 qcsapi_wifi_set_ap_isolate()

```
int qcsapi_wifi_set_ap_isolate (
            const char * ifname,
            const int new_ap_isolate )
```

Enable or disable AP isolation global control (applies across all BSSes).

When AP isolation is enabled, packets are not bridged between stations in the same BSS. When AP isolation is disabled, packets can be bridged between stations in the same BSS. AP isolation is disabled by default.

Note

This API can only be used in AP mode.

This API can only be used on the primary interface (wifi0)

When enabled, this API disables AP isolation on all BSSes, regardless of the individual interface configuration set using `set_intra_bss_isolate`

Parameters

ifname	the primary WiFi interface, wifi0 only.
new_ap_isolate	the new AP isolation setting

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_ap_isolate <WiFi interface> <0|1>
```

See also

qcsapi_ap_isolate_type

qcsapi_wifi_set_intra_bss_isolate

qcsapi_wifi_set_bss_isolate

9.8.6.103 qcsapi_wifi_get_intra_bss_isolate()

```
int qcsapi_wifi_get_intra_bss_isolate (
            const char * ifname,
            qcsapi_unsigned_int * p_ap_isolate )
```

This API returns the current intra-BSS isolation setting for the given interface.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_ap_isolate	return parameter to contain the the current intra-BSS isolation setting.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_intra_bss_isolate <WiFi interface>
```

Unless an error occurs, the output will be the current intra-BSS isolation setting for the given interface.

9.8.6.104 qcsapi_wifi_set_intra_bss_isolate()

```
int qcsapi_wifi_set_intra_bss_isolate (
            const char * ifname,
            const qcsapi_unsigned_int new_ap_isolate )
```

This API configures intra-BSS isolation. When enabled for a BSS, packets will not be forwarded from one station associated on the BSS to any other stations associated on the same BSS.

Note

This API can only be used in AP mode.

This setting is overridden by the global qcsapi_wifi_set_ap_isolate setting, if enabled.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
new_ap_isolate	the new intra-BSS isolation setting

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_intra_bss_isolate <WiFi interface> <0|1>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_ap_isolate_type

qcsapi_wifi_set_bss_isolate

qcsapi_wifi_set_ap_isolate

9.8.6.105 qcsapi_wifi_get_bss_isolate()

```
int qcsapi_wifi_get_bss_isolate (
            const char * ifname,
            qcsapi_unsigned_int * p_ap_isolate )
```

This API returns the current inter-BSS isolation setting for the given interface.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_ap_isolate	return parameter to contain the the current BSS isolation setting.

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_bss_isolate <WiFi interface>
```

Unless an error occurs, the output will be the current BSS isolation setting.

9.8.6.106 qcsapi_wifi_set_bss_isolate()

```
int qcsapi_wifi_set_bss_isolate (
            const char * ifname,
            const qcsapi_unsigned_int new_ap_isolate )
```

This API configures inter-BSS isolation. When enabled for a BSS, packets will not be forwarded from a station to any station associated on a different BSS.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
new_ap_isolate	the new BSS isolation setting

Returns

0 on success.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_bss_isolate <WiFi interface> <0|1>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_ap_isolate_type

qcsapi_wifi_set_ap_isolate

qcsapi_wifi_set_intra_bss_isolate

9.8.6.107 qcsapi_wifi_disable_dfs_channels()

```
int qcsapi_wifi_disable_dfs_channels (
            const char * ifname,
            int disable_dfs,
            int channel )
```

Configures the list of channels permitted during operation

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
disable_dfs	disable dfs channels
channel	channel to switch after dfs channels are disabled

Returns

0 on success, or other negative error codes on failure.

`call_qcsapi` interface:

```
call_qcsapi disable_dfs_channels wifi0 <0|1> [new channel]
```

9.8.6.108 qcsapi_wifi_is_ready()

```
int qcsapi_wifi_is_ready (
            qcsapi_unsigned_int * p_value )
```

This API call is used to retrieve the WiFi ready state

Parameters

p_value	the WiFi state, 1 is ready, 0 is not ready

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi is_wifi_ready <interface>
```

Unless an error occurs, the output will be 1 or 0.

9.8.6.109 qcsapi_wifi_test_traffic()

```
int qcsapi_wifi_test_traffic (
            const char * ifname,
            uint32_t period )
```

Start/Stop test traffic (null or qos null packets) on specified WiFi interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
period	Period interval for sending test traffic in milliseconds. 0 means disable.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi test_traffic <start|stop> <period>
```

9.8.6.110 qcsapi_wifi_add_multicast()

```
int qcsapi_wifi_add_multicast (
            qcsapi_unsigned_int ipaddr,
            qcsapi_mac_addr mac )
```

Add a static multicast entry to the forwarding table.

Parameters

ipaddr	the IP multicast address to be added to the table
mac	the MAC address of an associated station or wired interface

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

IGMP snooping should be disabled when using this command.

The MAC address must be present in the forwarding table. Using the MAC address of an associated station rather than a downstream endpoint is recommended because a downstream endpoint is only added to the forwarding table when traffic is received from it.

If a station disassociates, all multicast entries for the station are deleted.

Entries added by using this command are not aged out of the table and must be deleted by using `qcsapi_wifi_del_multicast`.

`call_qcsapi` interface:

```
call_qcsapi add_multicast <IP address> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.111 qcsapi_wifi_del_multicast()

```
int qcsapi_wifi_del_multicast (
            qcsapi_unsigned_int ipaddr,
            qcsapi_mac_addr mac )
```

Remove a multicast entry from the forwarding table.

Parameters

ipaddr	the IPv4 or IPv6 multicast address to be added to the table
mac	the MAC address of a known endpoint

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi del_multicast <IP address> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.112 qcsapi_wifi_get_multicast_list()

```
int qcsapi_wifi_get_multicast_list (
            char * buf,
            int buflen )
```

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_multicast_list
```

Unless an error occurs, the output is a list of multicast IP addresses and MAC addresses. If flood-forwarding is configured for a multicast IP address, the word 'flood' is printed in place of the MAC address list. E.g.

```
225.1.2.4 00:26:86:f0:32:d5 00:1b:21:71:78:e6 00:26:86:5c:16:7e
239.0.0.1 flood
225.1.2.3 00:26:86:5c:16:7e
...
```

9.8.6.113 qcsapi_wifi_add_ipff()

```
int qcsapi_wifi_add_ipff (
            qcsapi_unsigned_int ipaddr )
```

Add a multicast IP address to the flood-forwarding table. Packets matching these addresses will be flood-forwarded to every interface and every associated station.

Note

SSDP (239.255.255.250) and LNCB (224.0.0.0/24) packets are always flood-forwarded and do not need to be added to this table.

Parameters

ipaddr	the multicast IPv4 address to be added to the table

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi add_ipff <ipaddr>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.114 qcsapi_wifi_del_ipff()

```
int qcsapi_wifi_del_ipff (
            qcsapi_unsigned_int ipaddr )
```

Remove a multicast IP address from the flood-forwarding table.

Parameters

ipaddr	the multicast IPv4 address to be removed from the table

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi del_ipff <ipaddr>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.115 qcsapi_wifi_get_ipff()

```
int qcsapi_wifi_get_ipff (
            char * buf,
            int buflen )
```

Display the contents of the flood-forwarding table.

Note

SSDP (239.255.255.250) and LNCB (224.0.0.0/24) packets are always flood-forwarded, even if not added to this table.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_ipff
```

Unless an error occurs, the output is a list of configured multicast IP addresses, separated by newline characters.`complete`.

9.8.6.116 qcsapi_wifi_get_rts_threshold()

```
int qcsapi_wifi_get_rts_threshold (
            const char * ifname,
            qcsapi_unsigned_int * rts_threshold )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
rts_threshold	Output parameter to contain the value of RTS threshold

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rts_threshold <interface>
```

Unless an error occurs, outputs RTS threshold configured for interface

9.8.6.117 qcsapi_wifi_set_rts_threshold()

```
int qcsapi_wifi_set_rts_threshold (
            const char * ifname,
            qcsapi_unsigned_int rts_threshold )
```

Note

Value of RTS threshold should be in the range 0 - 65537; 0 - enables RTS/CTS for every frame, 65537 or more - disables RTS threshold

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
rts_threshold	New value of RTS threshold

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_rts_threshold <interface> <rts_threshold>
```

Unless an error occurs, the output will be the string `complete.`

9.8.6.118 qcsapi_wifi_set_nss_cap()

```
int qcsapi_wifi_set_nss_cap (
            const char * ifname,
            const qcsapi_mimo_type modulation,
            const qcsapi_unsigned_int nss )
```

This API call is used to set the maximum number of spatial streams for a given interface

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
modulation	either 'ht' (for 802.11n) or 'vht' (for 802.11ac)

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi set_nss_cap <WiFi interface> {ht | vht} <nss>
```

9.8.6.119 qcsapi_wifi_get_nss_cap()

```
int qcsapi_wifi_get_nss_cap (
            const char * ifname,
            const qcsapi_mimo_type modulation,
            qcsapi_unsigned_int * nss )
```

This API call is used to get the maximum number of spatial streams for a given interface

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
modulation	either 'ht' (for 802.11n) or 'vht' (for 802.11ac)

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi get_nss_cap <WiFi interface> {ht | vht}
```

9.8.6.120 qcsapi_wifi_get_tx_amsdu()

```
int qcsapi_wifi_get_tx_amsdu (
            const char * ifname,
            int * enable )
```

Parameters

ifname	\wifi 0
enable	returned A-MSDU status

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi get_tx_amsdu <WiFi interface>
```

9.8.6.121 qcsapi_wifi_set_tx_amsdu()

```
int qcsapi_wifi_set_tx_amsdu (
            const char * ifname,
            int enable )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable	0 to disable A-MSDU, 1 to enable it

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi set_tx_amsdu <WiFi interface> { 0 | 1 }
```

9.8.6.122 qcsapi_wifi_get_disassoc_reason()

```
int qcsapi_wifi_get_disassoc_reason (
            const char * ifname,
            qcsapi_unsigned_int * reason )
```

This API call is used to get disassoc reason.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi disassoc_reason <WiFi interface>
```

9.8.6.123 qcsapi_wifi_block_bss()

```
int qcsapi_wifi_block_bss (
            const char * ifname,
            const qcsapi_unsigned_int flag )
```

Block one BSS.

Parameters

ifname	wifix
flag	user configuration for the specified BSS. 1: Block 0 Unblock

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi block_bss <WiFi interface> <flag>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.124 qcsapi_wifi_get_block_bss()

```
int qcsapi_wifi_get_block_bss (
            const char * ifname,
            unsigned int * pvalue )
```

Get blocking status for association requests for one BSS.

Parameters

ifname	wifix
pvalue	blocking status. 1 - Blocked; 0 - Unblocked.

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_block_bss <WiFi interface>

Unless an error occurs, the output will be 0 or 1.  An error will give a
warning message.
```

9.8.6.125 qcsapi_wifi_verify_repeater_mode()

```
int qcsapi_wifi_verify_repeater_mode (
            void  )
```

Check if device is in repeater mode.

Returns

1 if in repeater mode.

0 if not in repeater mode.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

\call_qcsapi

```
call_qcsapi verify_repeater_mode
```

9.8.6.126 qcsapi_wifi_set_ap_interface_name()

```
int qcsapi_wifi_set_ap_interface_name (
            const char * ifname )
```

Configure the AP interface name for AP mode or primary AP interface name for repeater mode.

Note

The new name does not take effect until configuration is reloaded.

Parameters

ifname	the new interface name for AP mode

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

\call_qcsapi

```
call_qcsapi set_ap_interface_name <interface name>
```

9.8.6.127 qcsapi_wifi_get_ap_interface_name()

```
int qcsapi_wifi_get_ap_interface_name (
            char * ifname )
```

This API gets interface name for AP mode or primary AP interface name for repeater mode.

Parameters

ifname	the AP interface name returned

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

\call_qcsapi

```
call_qcsapi get_ap_interface_name
```

9.8.6.128 qcsapi_wifi_set_pref_band()

```
int qcsapi_wifi_set_pref_band (
            const char * ifname,
            const qcsapi_unsigned_int pref_band )
```

This API call is used to set preferred band between 2.4ghz or 5ghz.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

call_qcsapi interface:

```
call_qcsapi set_pref_band <WiFi interface> <2.4ghz | 5ghz>
```

9.8.6.129 qcsapi_wifi_get_pref_band()

```
int qcsapi_wifi_get_pref_band (
            const char * ifname,
            qcsapi_unsigned_int * pref_band )
```

This API call is used to get preferred band set between 2.4ghz or 5ghz.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0/1 on success, < 0 on error.

call_qcsapi interface:

```
call_qcsapi get_pref_band <WiFi interface>
```

Unless and error occurs, the output will be string 2.4ghz|5ghz .

```
In case of error output is "Please enter preferred band as 2.4ghz|5ghz" complete.
```

9.8.6.130 qcsapi_wifi_set_txba_disable()

```
int qcsapi_wifi_set_txba_disable (
            const char * ifname,
            const qcsapi_unsigned_int value )
```

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
disable	or enable TX BA

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

call_qcsapi interface:

```
call_qcsapi txba_disable <WiFi interface> <disable>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.131 qcsapi_wifi_get_txba_disable()

```
int qcsapi_wifi_get_txba_disable (
            const char * ifname,
            qcsapi_unsigned_int * value )
```

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
∗enable	Pointer to return buffer for storing the TX BA disable state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_txba_disable <WiFi interface>
```

Unless an error occurs, the output will be the TXBA disable status for the specific SSID.

9.8.6.132 qcsapi_wifi_set_rxba_decline()

```
int qcsapi_wifi_set_rxba_decline (
            const char * ifname,
            const qcsapi_unsigned_int value )
```

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
decline	or permit RX BA

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi rxba_decline <WiFi interface> <decline>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.133 qcsapi_wifi_get_rxba_decline()

```
int qcsapi_wifi_get_rxba_decline (
            const char * ifname,
            qcsapi_unsigned_int * value )
```

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
∗enable	Pointer to return buffer for storing the RX BA decline state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rxba_decline <WiFi interface>
```

Unless an error occurs, the output will be the RXBA decline status for the specific SSID.

9.8.6.134 qcsapi_wifi_set_txburst()

```
int qcsapi_wifi_set_txburst (
            const char * ifname,
            const qcsapi_unsigned_int enable )
```

Note

This API will not currently set the TX burst for the specified BSS. It is a stub only.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
enable	Enable or disable Tx burst

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_txburst <WiFi interface> <enable>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.135 qcsapi_wifi_get_txburst()

```
int qcsapi_wifi_get_txburst (
            const char * ifname,
            qcsapi_unsigned_int * enable )
```

Note

This API will always return 1 currently. It is a stub only.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
∗enable	Pointer to return buffer for storing the txburst state

Returns

0 if operation succeeds.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_txburst <WiFi interface>
```

Unless an error occurs, the output will be the tx burst status for the specified BSS.

9.8.6.136 qcsapi_wifi_get_sec_chan()

```
int qcsapi_wifi_get_sec_chan (
            const char * ifname,
            int chan,
            int * p_sec_chan )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
chan	assigned channel
p_sec_chan	returned secondary channel

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi get_sec_chan <WiFi interface> <chan>
```

9.8.6.137 qcsapi_wifi_set_sec_chan()

```
int qcsapi_wifi_set_sec_chan (
            const char * ifname,
            int chan,
            int offset )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
chan	assigned channel
offset	secondary channel offset, 0 means above and 1 means below

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi set_sec_chan <WiFi interface> <chan> <offset>
```

9.8.6.138 qcsapi_wifi_node_tx_airtime_accum_control()

```
int qcsapi_wifi_node_tx_airtime_accum_control (
            const char * ifname,
            const uint32_t node_index,
            qcsapi_airtime_control control )
```

This API configures per-node TX airtime accumulation start and stop

Parameters

ifname	interface name
node_index	node index
control	start or stop accumulation

Returns

0 if the command succeeded

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages

\call_qcsapi

```
call_qcsapi get_tx_airtime <interface name> <node_index | all> [start |
stop]
```

9.8.6.139 qcsapi_wifi_tx_airtime_accum_control()

```
int qcsapi_wifi_tx_airtime_accum_control (
            const char * ifname,
            qcsapi_airtime_control control )
```

This API configures TX airtime accumulation start and stop for all nodes

Parameters

ifname	interface name
control	start or stop accumulation

Returns

0 if the command succeeded

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages

\call_qcsapi

```
call_qcsapi get_tx_airtime <interface name> <node_index | all> [start |
stop]
```

9.8.6.140 qcsapi_wifi_node_get_txrx_airtime()

```
int qcsapi_wifi_node_get_txrx_airtime (
            const char * ifname,
            const uint32_t node_index,
            qcsapi_node_txrx_airtime * node_txrx_airtime )
```

This API gets per-node TX and RX airtime statistics

Parameters

ifname	interface name
node_index	node index
node_txrx_airtime	current airtime and cumulative airtime

Returns

0 if the command succeeded

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages

\call_qcsapi

```
call_qcsapi get_txrx_airtime <interface name> <node_index | all> [start |
stop]
```

9.8.6.141 qcsapi_wifi_get_txrx_airtime()

```
int qcsapi_wifi_get_txrx_airtime (
            const char * ifname,
            string_4096 buffer )
```

This API gets TX and RX airtime statistics for all nodes associated

Parameters

ifname	interface name
buffer	A pointer to the buffer for storing the returned value. The buffer is composed of fixed 2-byte nr_assoc_nodes, 2-byte free airtime and variable data, where variable data is nr_assoc_nodes * [2-byte node_idx, 6-byte node_mac, 4-byte airtime and 4-byte airtime_accum]

Returns

0 if the command succeeded

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages

\call_qcsapi

```
call_qcsapi get_txrx_airtime <interface name> <node_index | all> [start | stop]
```

9.8.6.142 qcsapi_wifi_set_max_bcast_pps()

```
int qcsapi_wifi_set_max_bcast_pps (
            const char * ifname,
            const qcsapi_unsigned_int max_bcast_pps )
```

This API call is used to limit the maximum number of broadcast packets transmitted to the BSS per second.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_bcast_pps	Maximum broadcast packets per second. The valid range is [0 - MAX_BCAST_PPS_LIMIT]. A value of 0 (disabled) will allow all broadcast packets through.

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi set_max_bcast_pps <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.8.6.143 qcsapi_wifi_is_weather_channel()

```
int qcsapi_wifi_is_weather_channel (
            const char * ifname,
            const uint16_t channel )
```

This API call is used to check if the channel provided is a weather one.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	the channel needs to check

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi is_weather_channel <WiFi interface> <channel>
```

Unless an error occurs, the output will be the string `1` or `0`.

9.8.6.144 qcsapi_wifi_get_tx_max_amsdu()

```
int qcsapi_wifi_get_tx_max_amsdu (
            const char * ifname,
            int * max_len )
```

Get the configured maximum A-MSDU size.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_len	buffer for the returned value

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi get_tx_max_amsdu <WiFi interface>
```

9.8.6.145 qcsapi_wifi_set_tx_max_amsdu()

```
int qcsapi_wifi_set_tx_max_amsdu (
            const char * ifname,
            int max_len )
```

Set the maximum A-MSDU size.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_len	0, 1 or 2 to set the max Tx A-MSDU size to 4kB, 8kB or 11kB, respectively

Returns

0 on success or a negative value on error.

\call_qcsapi

```
call_qcsapi set_tx_max_amsdu <WiFi interface> <max_len>
```

9.9 MBSSID APIs

MBSSID is a feature that allows additional AP-mode virtual interfaces to be configured on a single device. Each additional virtual interface is created as a new network interface, so existing security APIs and generic interface APIs can be used on the new interface.

Functions

- int qcsapi_wifi_create_restricted_bss (const char *ifname, const qcsapi_mac_addr mac_addr)

  Create a new restricted BSS.
- int qcsapi_wifi_create_bss (const char *ifname, const qcsapi_mac_addr mac_addr)

  Create a new BSS.
- int qcsapi_wifi_remove_bss (const char *ifname)

  Remove a BSS.

9.9.1 Detailed Description

Note

All MBSSID APIs work with the host AP daemon security configuration file, hostapd.conf. Its location is determined by the get file path configuration API. Results from these APIs may be inconsistent or incorrect if the file path to the security configuration files has not been correctly configured.

9.9.2 Function Documentation

9.9.2.1 qcsapi_wifi_create_restricted_bss()

```
int qcsapi_wifi_create_restricted_bss (
            const char * ifname,
            const qcsapi_mac_addr mac_addr )
```

Creates a new MBSSID AP-mode virtual interface with a set of default security parameters. This new virtual interface isn't added into back-end bridge, just for test.

After calling this API function the host AP security daemon configuration is updated, reloaded and the new BSS is made active. Subsequent security and general interface APIs can then be called on the new virtual interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mac_addr	mac address of the created vap. If set to 0, driver will generate one mac address based on primary interface mac address.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wifi_create_restricted_bss <WiFi interface> [mac_addr]
```

Unless an error occurs, the output will be the string `complete`.

9.9.2.2 qcsapi_wifi_create_bss()

```
int qcsapi_wifi_create_bss (
            const char * ifname,
            const qcsapi_mac_addr mac_addr )
```

Creates a new MBSSID AP-mode virtual interface with a set of default security parameters.

After calling this API function the host AP security daemon configuration is updated, reloaded and the new BSS is made active. Subsequent security and general interface APIs can then be called on the new virtual interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0). not be present in the hostapd.conf file.
mac_addr	mac address of the created vap. If set to 0, driver will generate one mac address based on primary interface mac address.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wifi_create_bss <WiFi interface> [mac_addr]
```

Unless an error occurs, the output will be the string `complete`.

9.9.2.3 qcsapi_wifi_remove_bss()

```
int qcsapi_wifi_remove_bss (
            const char * ifname )
```

Removes an existing MBSSID AP-mode virtual interface with interface name ifname.

The API will return an error if the named interface does not exist.

After calling this function, the host AP security daemon configuration is modified, reloaded and the interface named is no longer active.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wifi_remove_bss <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.10 WDS APIs

WDS (Wireless Distribution System) here means a link between two APs on the same 802.11 channel, between which traffic is allowed to flow using 4 address frames.
It is important to note that the implementation of WDS peering is not an official 802.11 standard. 802.11 only details the implementation of 4 address frames, not any of the setup or negotiation between peers.
The implementation is largely targeted at interoperability with units produced by a specific customer using a competitor's chipset.
A WDS peer is identified by its primary BSSID. Recall a BSSID is represented as a MAC address, in this context the MAC address of the peer AP.
The WDS peering agreement is symmetric. Both sides need to have the peer address of the other added. Otherwise no WDS connection will be established. If only one side of a WDS link has added the other, the peer AP will not recognize the connection.

Functions

- int qcsapi_wds_add_peer (const char ∗ifname, const qcsapi_mac_addr peer_address)

  Add a WDS peer.
- int qcsapi_wds_add_peer_encrypt (const char ∗ifname, const qcsapi_mac_addr peer_address, const qcsapi_unsigned_int encryption)

  Add a WDS peer with an encrypted link.
- int qcsapi_wds_remove_peer (const char ∗ifname, const qcsapi_mac_addr peer_address)

  Remove a WDS peer.
- int qcsapi_wds_get_peer_address (const char ∗ifname, const int index, qcsapi_mac_addr peer_address)

  Get a WDS peer by index.
- int qcsapi_wds_set_psk (const char ∗ifname, const qcsapi_mac_addr peer_address, const string_64 pre_↩ shared_key)

  Set the WPA PSK for a WDS peer connection.
- int qcsapi_wds_set_mode (const char ∗ifname, const qcsapi_mac_addr peer_address, const int mode)

  Set the WDS mode for a WDS peer connection.
- int qcsapi_wds_get_mode (const char ∗ifname, const int index, int ∗mode)

  Get a WDS peer mode by index.
- int qcsapi_wifi_set_extender_params (const char ∗ifname, const qcsapi_extender_type type, const int param_value)

  Set Extender device parameter.
- int qcsapi_wifi_get_extender_params (const char ∗ifname, const qcsapi_extender_type type, int ∗p_value)

  get all Extender device parameters infomation

9.10.1 Detailed Description

9.10.2 Function Documentation

9.10.2.1 qcsapi_wds_add_peer()

```
int qcsapi_wds_add_peer (
            const char * ifname,
            const qcsapi_mac_addr peer_address )
```

This API adds a new WDS peer with the given peer address.

An error is returned if the maximum number of WDS peers has been reached or if the peer address already exists.

Note

The ifname parameter must refer to a primary interface. All WDS peers belong to the primary interface only. This API allows unencrypted data to be transmitted across the WDS link as soon as it has been established. If the link will be encrypted, use `qcsapi_wds_add_peer_encrypt` instead.

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
peer_address	the peer address to add to the WDS interface.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_add_peer <WiFi interface> <BSSID of peer AP>
```

Section Format for a MAC address shows how to format the BSSID (MAC address).

Unless an error occurs, the output will be the string `complete`.

9.10.2.2 qcsapi_wds_add_peer_encrypt()

```
int qcsapi_wds_add_peer_encrypt (
            const char * ifname,
            const qcsapi_mac_addr peer_address,
            const qcsapi_unsigned_int encryption )
```

This API adds a new WDS peer with the given peer address.

An error is returned if the maximum number of WDS peers has been reached or if the peer address already exists.

Note

This API can only be used on the primary interface (wifi0)

The ifname parameter must refer to a primary interface. All WDS peers belong to the primary interface only.

Parameters

ifname	the primary WiFi interface, wifi0 only.
peer_address	the peer address to add to the WDS interface
encryption	0 if the link will not be encrypted or 1 if the link will be encrypted

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_add_peer <WiFi interface> <BSSID of peer AP> [encrypt]
```

Section Format for a MAC address shows how to format the BSSID (MAC address). If `encrypt` is set, no data is transmitted across the WDS link until it has been secured by using `call_qcsapi wds_set_psk`.

Note

Not enabling encryption means that data can be transmitted across the WDS link as soon as it has been configured but does not preclude encrypting the link.

Unless an error occurs, the output will be the string `complete`.

9.10.2.3 qcsapi_wds_remove_peer()

```
int qcsapi_wds_remove_peer (
            const char * ifname,
            const qcsapi_mac_addr peer_address )
```

This API removes an existing WDS peer. An error is returned if the peer does not exist.

Note

This API can only be used on the primary interface (wifi0)

The ifname parameter must refer to a primary interface. All WDS peers belong to the primary interface only.

Parameters

ifname	the primary WiFi interface, wifi0 only.
peer_address	the peer address to remove from the WDS interface.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_remove_peer <WiFi interface> <BSSID of peer AP>
```

Section Format for a MAC address shows how to format the BSSID (MAC address).

Unless an error occurs, the output will be the string `complete`.

9.10.2.4 qcsapi_wds_get_peer_address()

```
int qcsapi_wds_get_peer_address (
            const char * ifname,
            const int index,
            qcsapi_mac_addr peer_address )
```

This API is used to find a WDS peer address by index.

This API is typically used to construct a list of the configured WDS peers.

Note

This API can only be used on the primary interface (wifi0)

The ifname parameter must refer to a primary interface. All WDS peers belong to the primary interface only.

Parameters

ifname	the primary WiFi interface, wifi0 only.
index	the index to get the WDS peer address of.
peer_address	return parameter to contain the peer address of the given index.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_get_peer_address <WiFi interface> <index>
```

Section Format for a MAC address shows how to format the BSSID (MAC address).

Unless an error occurs, the output will be the BSSID for the peer, as selected by its index.

9.10.2.5 qcsapi_wds_set_psk()

```
int qcsapi_wds_set_psk (
            const char * ifname,
            const qcsapi_mac_addr peer_address,
            const string_64 pre_shared_key )
```

The WDS link between two APs in a pair can be encrypted. This encryption is per-peer.

The scheme used for WDS encryption is similar to WPA-NONE encryption which is often used for ad-hoc connections. There is no key exchange protocol. Frames are encrypted with AES using a 256-bit preshared key, set to the same value on both peers.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
peer_address	the WDS peer to apply the key to.
pre_shared_key	a 256 bit key, encoded as hexadecimal ASCII characters (0-9, a-f). If this parameter is NULL, the WDS peer key will be cleared.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_set_psk <WiFi interface> <BSSID of peer AP> <WDS PSK>
```

where <WiFi interface> is the primary interface, <BSSID of peer AP> is the BSSID of the peer AP and <WDS PSK> is the PSK. Enter the PSK as a string of exactly 64 hexadecimal digits, or NULL to pass an empty PSK to the API.

Unless an error occurs, the output will be the string `complete`.

9.10.2.6 qcsapi_wds_set_mode()

```
int qcsapi_wds_set_mode (
            const char * ifname,
            const qcsapi_mac_addr peer_address,
            const int mode )
```

The WDS peer can play a role of either MBS (main base station) or RBS (remote base station).

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
peer_address	the WDS peer to apply the mode to.
mode	non-zero value for rbs mode and zero for mbs mode.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_set_mode <WiFi interface> <BSSID of peer AP> <mode string>
```

where <WiFi interface> is the primary interface, <BSSID of peer AP> is the BSSID of the peer AP and <mode string> is either "rbs" or "mbs".

Unless an error occurs, the output will be the string `complete`.

9.10.2.7 qcsapi_wds_get_mode()

```
int qcsapi_wds_get_mode (
            const char * ifname,
            const int index,
            int * mode )
```

This API is used to find a WDS peer mode by index.

Note

This API can only be used on the primary interface (wifi0)

The ifname parameter must refer to a primary interface. All WDS peers belong to the primary interface only.

Parameters

ifname	the primary WiFi interface, wifi0 only.
index	the index to get the WDS peer address of.
mode	return parameter to contain the peer mode of the given index.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wds_get_mode <WiFi interface> <index>
```

Unless an error occurs, the output will be the mode for the peer, as selected by its index.

9.10.2.8 qcsapi_wifi_set_extender_params()

```
int qcsapi_wifi_set_extender_params (
            const char * ifname,
            const qcsapi_extender_type type,
            const int param_value )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	Extender parameter type
param_value	Extender parameter value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_extender_params <WiFi interface> <parameter type> <parameter value>
```

where `WiFi interface` is the primary interface, `parameter type` is one of `role`, `mbs_best_rssi`, `rbs_best_rssi`, `mbs_wgt`, `rbs_wgt`, `roaming`, `bgscan_interval`, `verbose`, `mbs_rssi_margin`.

Unless an error occurs, the output will be the string `complete`.

9.10.2.9 qcsapi_wifi_get_extender_params()

```
int qcsapi_wifi_get_extender_params (
            const char * ifname,
            const qcsapi_extender_type type,
            int * p_value )
```

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
type	Extender parameter type
p_value	Extender parameter value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_extender_status <WiFi interface>
```

where `WiFi interface` is the primary interface, Unless an error occurs, the output will be the Extender related parameter value.

9.11 Security APIs

These APIs are for the Access Point (AP) and do not work on a Station (STA).

Functions

- int qcsapi_wifi_get_beacon_type (const char *ifname, char *p_current_beacon)

  Get the security protocol from the beacon.
- int qcsapi_wifi_set_beacon_type (const char *ifname, const char *p_new_beacon)

  Set the security protocol in the beacon.
- int qcsapi_wifi_get_WEP_key_index (const char *ifname, qcsapi_unsigned_int *p_key_index)

  API is not supported.
- int qcsapi_wifi_set_WEP_key_index (const char *ifname, const qcsapi_unsigned_int key_index)

  API is not supported.
- int qcsapi_wifi_get_WEP_key_passphrase (const char *ifname, string_64 current_passphrase)

  API is not supported.
- int qcsapi_wifi_set_WEP_key_passphrase (const char *ifname, const string_64 new_passphrase)

  API is not supported.
- int qcsapi_wifi_get_WEP_encryption_level (const char *ifname, string_64 current_encryption_level)

  Retrieves current encryption level and supported encryption levels.
- int qcsapi_wifi_get_basic_encryption_modes (const char *ifname, string_32 encryption_modes)

  API is not supported.
- int qcsapi_wifi_set_basic_encryption_modes (const char *ifname, const string_32 encryption_modes)

  API is not supported.
- int qcsapi_wifi_get_basic_authentication_mode (const char *ifname, string_32 authentication_mode)

  API is not supported.
- int qcsapi_wifi_set_basic_authentication_mode (const char *ifname, const string_32 authentication_mode)

  API is not supported.
- int qcsapi_wifi_get_WEP_key (const char *ifname, qcsapi_unsigned_int key_index, string_64 current_↩ passphrase)

  API is not supported.
- int qcsapi_wifi_set_WEP_key (const char *ifname, qcsapi_unsigned_int key_index, const string_64 new_↩ passphrase)

  API is not supported.
- int qcsapi_wifi_get_WPA_encryption_modes (const char *ifname, string_32 encryption_modes)

  *Get the security **encryption** mode(s) configured.*
- int qcsapi_wifi_set_WPA_encryption_modes (const char *ifname, const string_32 encryption_modes)

  *Set the security **encryption** mode(s).*
- int qcsapi_wifi_get_WPA_authentication_mode (const char *ifname, string_32 authentication_mode)

  *Get the security **authentication** mode configured.*
- int qcsapi_wifi_set_WPA_authentication_mode (const char *ifname, const string_32 authentication_mode)

  *Set the security **authentication** mode.*
- int qcsapi_wifi_get_interworking (const char *ifname, string_32 interworking)

  Get the 802.11u Interworking status.
- int qcsapi_wifi_set_interworking (const char *ifname, const string_32 interworking)

  Set the 802.11u Interworking status.
- int qcsapi_wifi_get_80211u_params (const char *ifname, const string_32 u_param, string_256 p_buffer)

  Get an 802.11u parameter.
- int qcsapi_wifi_set_80211u_params (const char *ifname, const string_32 param, const string_256 value1, const string_32 value2)

Set an 802.11u parameter.

- int qcsapi_security_get_nai_realms (const char *ifname, string_4096 p_value)

  Get 802.11 NAI Realms.

- int qcsapi_security_add_nai_realm (const char *ifname, const int encoding, const char *nai_realm, const char *eap_method)

  Add or update an 802.11 NAI Realm.

- int qcsapi_security_del_nai_realm (const char *ifname, const char *nai_realm)

  Delete an 802.11u NAI Realm.

- int qcsapi_security_get_roaming_consortium (const char *ifname, string_1024 p_value)

  Get 802.11u Roaming Consortia.

- int qcsapi_security_add_roaming_consortium (const char *ifname, const char *p_value)

  Add the 802.11u roaming_consortium.

- int qcsapi_security_del_roaming_consortium (const char *ifname, const char *p_value)

  Delete a 802.11u Roaming Consortium.

- int qcsapi_security_get_venue_name (const char *ifname, string_4096 p_value)

  Get 802.11u Venue names.

- int qcsapi_security_add_venue_name (const char *ifname, const char *lang_code, const char *venue_name)

  Add the 802.11u venue name.

- int qcsapi_security_del_venue_name (const char *ifname, const char *lang_code, const char *venue_name)

  Delete the 802.11u venue name.

- int qcsapi_security_get_oper_friendly_name (const char *ifname, string_4096 p_value)

  Get Hotspot 2.0 opererator friendly names.

- int qcsapi_security_add_oper_friendly_name (const char *ifname, const char *lang_code, const char *oper_friendly_name)

  Add Hotspot 2.0 opererator friendly name.

- int qcsapi_security_del_oper_friendly_name (const char *ifname, const char *lang_code, const char *oper_friendly_name)

  Delete Hotspot 2.0 opererator friendly name.

- int qcsapi_security_get_hs20_conn_capab (const char *ifname, string_4096 p_value)

  Get Hotspot 2.0 connection capability.

- int qcsapi_security_add_hs20_conn_capab (const char *ifname, const char *ip_proto, const char *port_num, const char *status)

  Add Hotspot 2.0 connection capability.

- int qcsapi_security_del_hs20_conn_capab (const char *ifname, const char *ip_proto, const char *port_num, const char *status)

  Delete Hotspot 2.0 connection capability.

- int qcsapi_wifi_get_hs20_status (const char *ifname, string_32 p_hs20)

  Get the Hotspot 2.0 parameter status.

- int qcsapi_wifi_set_hs20_status (const char *ifname, const string_32 hs20_val)

  Enable or Disable Hotspot 2.0.

- int qcsapi_wifi_get_proxy_arp (const char *ifname, string_32 p_proxy_arp)

  Get the Proxy ARP parameter status.

- int qcsapi_wifi_set_proxy_arp (const char *ifname, const string_32 proxy_arp_val)

  Set the Proxy ARP parameter.

- int qcsapi_wifi_get_l2_ext_filter (const char *ifname, const string_32 param, string_32 value)

  Get the L2 external filter parameters.

- int qcsapi_wifi_set_l2_ext_filter (const char *ifname, const string_32 param, const string_32 value)

  Set the L2 external filter parameters.

- int qcsapi_wifi_get_hs20_params (const char *ifname, const string_32 hs_param, string_32 p_buffer)

  Get a Hotspot 2.0 parameter value.

- int qcsapi_wifi_set_hs20_params (const char *ifname, const string_32 hs_param, const string_64 value1, const string_64 value2, const string_64 value3, const string_64 value4, const string_64 value5, const string_64 value6)

  Set a Hotspot 2.0 parameter value.
- int qcsapi_remove_11u_param (const char *ifname, const string_64 param)

  Remove the 802.11u parameter.
- int qcsapi_remove_hs20_param (const char *ifname, const string_64 hs_param)

  Remove a Hotspot 2.0 parameter.
- int qcsapi_security_add_hs20_icon (const char *ifname, const qcsapi_unsigned_int icon_width, const qcsapi_unsigned_int icon_height, const char *lang_code, const char *icon_type, const char *icon_name, const char *file_path)

  Add a Hotspot 2.0 icon description.
- int qcsapi_security_get_hs20_icon (const char *ifname, string_1024 value)

  Get all Hotspot 2.0 icon descriptions.
- int qcsapi_security_del_hs20_icon (const char *ifname, const string_1024 icon_name)

  Delete a Hotspot 2.0 icon description.
- int qcsapi_security_add_osu_server_uri (const char *ifname, const string_256 osu_server_uri)

  Add OSU Provider server URI.
- int qcsapi_security_get_osu_server_uri (const char *ifname, string_1024 value)

  Get all OSU servers URIs.
- int qcsapi_security_del_osu_server_uri (const char *ifname, const string_256 osu_server_uri)

  Delete OSU Provider.
- int qcsapi_security_add_osu_server_param (const char *ifname, const string_256 osu_server_uri, const string_256 param, const string_256 value)

  Add an OSU Provider server parameter.
- int qcsapi_security_get_osu_server_param (const char *ifname, const string_256 osu_server_uri, const string_256 param, string_1024 value)

  Get values for OSU server parameter.
- int qcsapi_security_del_osu_server_param (const char *ifname, const string_256 osu_server_uri, const string_256 param, const string_256 value)

  Delete an OSU Provider parameter.
- int qcsapi_wifi_get_IEEE11i_encryption_modes (const char *ifname, string_32 encryption_modes)

  see qcsapi_wifi_get_WPA_encryption_modes
- int qcsapi_wifi_set_IEEE11i_encryption_modes (const char *ifname, const string_32 encryption_modes)

  see qcsapi_wifi_set_WPA_encryption_modes
- int qcsapi_wifi_get_IEEE11i_authentication_mode (const char *ifname, string_32 authentication_mode)

  see qcsapi_wifi_get_WPA_authentication_mode
- int qcsapi_wifi_set_IEEE11i_authentication_mode (const char *ifname, const string_32 authentication_↩ mode)

  see qcsapi_wifi_set_WPA_authentication_mode
- int qcsapi_wifi_get_michael_errcnt (const char *ifname, uint32_t *errcount)

  Get TKIP MIC errors count.
- int qcsapi_wifi_get_pre_shared_key (const char *ifname, const qcsapi_unsigned_int key_index, string_64 pre_shared_key)

  Get the preshared key.
- int qcsapi_wifi_set_pre_shared_key (const char *ifname, const qcsapi_unsigned_int key_index, const string_64 pre_shared_key)

  Set the preshared key.
- int qcsapi_wifi_add_radius_auth_server_cfg (const char *ifname, const char *radius_auth_server_ipaddr, const char *radius_auth_server_port, const char *radius_auth_server_sh_key)

  Add RADIUS authentication server.

- int qcsapi_wifi_del_radius_auth_server_cfg (const char *ifname, const char *radius_auth_server_ipaddr, const char *constp_radius_port)

  Remove RADIUS authentication server.
- int qcsapi_wifi_get_radius_auth_server_cfg (const char *ifname, string_1024 radius_auth_server_cfg)

  Get RADIUS authentication servers.
- int qcsapi_wifi_set_own_ip_addr (const char *ifname, const string_16 own_ip_addr)

  Set the EAP own ip address of the AP.
- int qcsapi_wifi_get_key_passphrase (const char *ifname, const qcsapi_unsigned_int key_index, string_64 passphrase)

  Get the WiFi passphrase for the given interface.
- int qcsapi_wifi_set_key_passphrase (const char *ifname, const qcsapi_unsigned_int key_index, const string_64 passphrase)

  Set the WiFi passphrase (ASCII) for the given interface.
- int qcsapi_wifi_get_group_key_interval (const char *ifname, unsigned int *p_key_interval)

  Get the group key rotation interval.
- int qcsapi_wifi_get_pairwise_key_interval (const char *ifname, unsigned int *p_key_interval)

  Get the pairwise key rotation interval.
- int qcsapi_wifi_set_group_key_interval (const char *ifname, const unsigned int key_interval)

  Set the group key rotation interval.
- int qcsapi_wifi_set_pairwise_key_interval (const char *ifname, const unsigned int key_interval)

  Set the pairwise key rotation interval.
- int qcsapi_wifi_get_pmf (const char *ifname, int *p_pmf_cap)

  Get the 802.11w capability for the given interface.
- int qcsapi_wifi_set_pmf (const char *ifname, int pmf_cap)

  Set the 802.11w / PMF capability for the given interface.
- int qcsapi_wifi_get_wpa_status (const char *ifname, char *wpa_status, const char *mac_addr, const qcsapi_unsigned_int max_len)

  Get the the WPA status for the given interface.
- int qcsapi_wifi_get_psk_auth_failures (const char *ifname, qcsapi_unsigned_int *count)

  Get the total number of PSK authentication failures.
- int qcsapi_wifi_get_auth_state (const char *ifname, const char *mac_addr, int *auth_state)

  Get the the authenticated state of the specific station according to the mac_addr for the given interface.
- int qcsapi_wifi_set_security_defer_mode (const char *ifname, int defer)

  Set security defer mode.
- int qcsapi_wifi_get_security_defer_mode (const char *ifname, int *defer)

  Get security defer mode.
- int qcsapi_wifi_apply_security_config (const char *ifname)

  Apply security config.

9.11.1 Detailed Description

For the equivalent STA APIs, see section SSID APIs. The interface parameter must reference either the primary interface or a previously created AP-mode virtual interface (MBSSID feature).

Note

All Security APIs work with the host AP daemon security configuration file, hostapd.conf. Its location is determined by the get file path configuration API (section File Path configuration). Results from these APIs may be inconsistent or incorrect if the file path to the security configuration files has not been correctly configured.

9.11.2 Security definitions

The following table outlines the defined string **authentication protocols** as used commonly throughout the QCS↩ API.

Value	Interpretation
Basic	No security in use
WPA	WPA version 1 authentication protocol
11i	802.11i authentication protocol
WPAand11i	Both WPA and 802.11i authentication protocols are available

The following table outlines the defined string **encryption** types as used commonly throughout the QCSAPI.

Value	Interpretation
AESEncryption	AES(CCMP) Encryption in use.
TKIPEncryption	TKIP Encryption in use.
TKIPandAESEncryption	TKIP and AES(CCMP) Encryption in use.

The following table outlines the defined string **authentication types** as used commonly througout the QCSAPI.

Value	Interpretation
PSKAuthentication	Pre-shared key authentication.
EAPAuthentication	Use of an EAP server for authentication.

9.11.3 Authentication protocols and encryption

This section has a few sentences to try and clarify the difference between authentication and encryption, and the different methods as documented in the previous tables. All are closely inter-related, but are different parts of the same stick.

Authentication is the act of verifying an entity is allowed access to a resource. In the case of 802.11 devices, authentication is done through one of multiple methods:

- NULL authentication (eg, OPEN networks) - "None"
- Pre-shared WEP key (obsolete - not mentioned further).
- Pre-shared key with WPA/WPA2 authentication - "WPA" or "11i" or "WPAand11i" - collectively "PSK↩ Authentication"
- Extensible Authentication Protocol (EAP). - EAP-FAST, EAP-PEAP, ... - collectively "EAPAuthentication"

The Quantenna software implicitly supports "None" and "PSKAuthentication". "EAPAuthentication" can be added by customers, as it sits at the userspace level.

Typically, once authentication has completed, one of the outputs from the authentication protocol will be a set of temporary keys.

These keys are then used for the second part of the security equation, for **encryption**.

Encryption takes plain text (or packets) and applies a cryptographic algorithm, using a known (or derived) shared key, to generate encrypted text (or packets). The different algorithms used for encryption are negotiated during initial connection establishment, and the supported encryption algorithms are:

- NONE (no encryption)

- TKIP (Rotating WEP or RC4 key)
- CCMP (or AES key)

Generally speaking, devices using an encryption key will have two keys - one for unicast (AP->STA and STA->AP), and one for broadcast and multicast (AP-> all STAs in the BSS).

Functions within the QCSAPI that deal with security have three broad categories for defining the security setup:

- Setting the authentication type (eg, PSK, EAP, NONE, etc.)
- Setting the specific authentication protocol (eg, WPA, WPA2, PEAP etc.)
- Setting the encryption type (eg, TKIP, CCMP)

The following table shows the different functions used for these different tasks - both set and get functions.

Function	Get	Set
Get/set authentication type	qcsapi_wifi_get_WPA_authentication_mode, qcsapi_SSID_get_authentication_mode	qcsapi_wifi_set_WPA_authentication_mode, qcsapi_SSID_set_authentication_mode
Get/set authentication protocol	qcsapi_wifi_get_beacon_type, qcsapi_SSID_get_protocol	qcsapi_wifi_set_beacon_type, qcsapi_SSID_set_protocol
Get/set encryption type	qcsapi_wifi_get_WPA_encryption_modes, qcsapi_SSID_get_encryption_modes	qcsapi_wifi_set_WPA_encryption_modes, qcsapi_SSID_set_encryption_modes

9.11.4 Function Documentation

9.11.4.1 qcsapi_wifi_get_beacon_type()

```
int qcsapi_wifi_get_beacon_type (
            const char * ifname,
            char * p_current_beacon )
```

Get the current beacon type. Only applicable for an AP; for a STA, use the SSID Get Protocol API (qcsapi_SSID_get_protocol). On success, returned string will be one of those as documented in the **authentication protocol** table in Security definitions

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_current_beacon	the protocol as returned by the API.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_beacon <WiFi interface>
```

Unless an error occurs, the response will be one of Basic, 11i, WPA or WPAand11i with the interpretation of each listed in the table in the **authentication protocol** table in Security definitions.

See also

qcsapi_SSID_get_protocol

9.11.4.2 qcsapi_wifi_set_beacon_type()

```
int qcsapi_wifi_set_beacon_type (
            const char * ifname,
            const char * p_new_beacon )
```

Set the current beacon type.

This API only applies for an AP; for a Station, use the SSID Set Protocol API (qcsapi_SSID_set_protocol).

The value for the new beacon must be one of the expected values listed in the section on the corresponding get API. Value must match exactly including upper vs. lower case letters.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_new_beacon	the new security protocol to set in the beacon.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_beacon <WiFi interface> <beacon type>
```

Unless an error occurs, the output will be the string `complete`.

Beacon type needs to be one of the values as per the **authentication protocol** table in Security definitions.

See also

qcsapi_SSID_set_protocol

9.11.4.3 qcsapi_wifi_get_WEP_key_index()

```
int qcsapi_wifi_get_WEP_key_index (
            const char * ifname,
            qcsapi_unsigned_int * p_key_index )
```

Returns

-EOPNOTSUPP.

9.11.4.4 qcsapi_wifi_set_WEP_key_index()

```
int qcsapi_wifi_set_WEP_key_index (
            const char * ifname,
            const qcsapi_unsigned_int key_index )
```

Returns

-EOPNOTSUPP.

9.11.4.5 qcsapi_wifi_get_WEP_key_passphrase()

```
int qcsapi_wifi_get_WEP_key_passphrase (
            const char * ifname,
            string_64 current_passphrase )
```

Returns

-EOPNOTSUPP.

9.11.4.6 qcsapi_wifi_set_WEP_key_passphrase()

```
int qcsapi_wifi_set_WEP_key_passphrase (
            const char * ifname,
            const string_64 new_passphrase )
```

Returns

-EOPNOTSUPP.

9.11.4.7 qcsapi_wifi_get_WEP_encryption_level()

```
int qcsapi_wifi_get_WEP_encryption_level (
            const char * ifname,
            string_64 current_encryption_level )
```

qcsapi_wifi_get_WEP_encryption_level return current encryption level describing current encryption state and available encrytion options for example 'Disabled,40-bit,104-bit,128-bit'

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_encryption_level	String to store encryption level data. Type string string_64

Returns

>= 0 on success

-EFAULT if current_encryption_level is NULL

-EMSGSIZE or negative number if underlying wireless_extensions API indicated an error

`call_qcsapi` interface:

```
call_qcsapi get_WEP_encryption_level <WiFi interface> <current_encryption↩
_level>
```

Unless an error occurs, the output will be the string describing the encryption level for example 'Disabled,40-bit,104-bit,12

9.11.4.8 qcsapi_wifi_get_basic_encryption_modes()

```
int qcsapi_wifi_get_basic_encryption_modes (
            const char * ifname,
            string_32 encryption_modes )
```

Returns

-EOPNOTSUPP.

9.11.4.9 qcsapi_wifi_set_basic_encryption_modes()

```
int qcsapi_wifi_set_basic_encryption_modes (
            const char * ifname,
            const string_32 encryption_modes )
```

Returns

-EOPNOTSUPP.

9.11.4.10 qcsapi_wifi_get_basic_authentication_mode()

```
int qcsapi_wifi_get_basic_authentication_mode (
            const char * ifname,
            string_32 authentication_mode )
```

Returns

-EOPNOTSUPP.

9.11.4.11 qcsapi_wifi_set_basic_authentication_mode()

```
int qcsapi_wifi_set_basic_authentication_mode (
            const char * ifname,
            const string_32 authentication_mode )
```

Returns

-EOPNOTSUPP.

9.11.4.12 qcsapi_wifi_get_WEP_key()

```
int qcsapi_wifi_get_WEP_key (
            const char * ifname,
            qcsapi_unsigned_int key_index,
            string_64 current_passphrase )
```

Returns

-EOPNOTSUPP

9.11.4.13 qcsapi_wifi_set_WEP_key()

```
int qcsapi_wifi_set_WEP_key (
            const char * ifname,
            qcsapi_unsigned_int key_index,
            const string_64 new_passphrase )
```

Returns

-EOPNOTSUPP

9.11.4.14 qcsapi_wifi_get_WPA_encryption_modes()

```
int qcsapi_wifi_get_WPA_encryption_modes (
            const char * ifname,
            string_32 encryption_modes )
```

Get the current WPA/11i encryption protocol(s) in use. Applies to AP only. For a STA, use qcsapi_SSID_get_↩ encryption_modes.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
encryption_modes	a string containing a value per the **encryption** definitions table in the Security definitions section.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_WPA_encryption_modes <WiFi interface>
```

Unless an error occurs, the output will be one of the strings listed in the **encryption** definitions table in the Security definitions section.

See also

qcsapi_SSID_get_encryption_modes

9.11.4.15 qcsapi_wifi_set_WPA_encryption_modes()

```
int qcsapi_wifi_set_WPA_encryption_modes (
            const char * ifname,
            const string_32 encryption_modes )
```

Set the current security encryption mode(s). Applies to AP only. For a STA, use qcsapi_SSID_set_encryption_↩ modes. Value is required to be one of the expected values from the corresponding get operation. Value must match exactly including upper vs. lower case letters.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
encryption_modes	a string containing a value per the **encryption** definitions table in the Security definitions section.

`call_qcsapi` interface:

```
call_qcsapi set_WPA_encryption_modes <WiFi interface> <encryption mode(s)>
```

Unless an error occurs, the output will be the string `complete`.

Encryptions mode(s) needs to be one of the values per the **encryption** definitions table in the Security definitions section.

See also

qcsapi_SSID_set_encryption_modes

9.11.4.16 qcsapi_wifi_get_WPA_authentication_mode()

```
int qcsapi_wifi_get_WPA_authentication_mode (
            const char * ifname,
            string_32 authentication_mode )
```

Get the current security authentication mode in use. Applies to AP only. For a STA, use qcsapi_SSID_get_↩ authentication_mode

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
authentication_mode	a string containing a value per the **authentication types** table in the Security definitions section.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_WPA_authentication_mode <WiFi interface>
```

Unless an error occurs, the output will be one of the strings listed in the **authentication type** table in the Security definitions section.

See also

qcsapi_SSID_get_authentication_mode

9.11.4.17 qcsapi_wifi_set_WPA_authentication_mode()

```
int qcsapi_wifi_set_WPA_authentication_mode (
            const char * ifname,
            const string_32 authentication_mode )
```

Set the current security authentication mode. Applies to AP only. For a STA, use qcsapi_SSID_set_authentication↩ _modes. Value is required to be one of the expected values from the corresponding get operation. Value must match exactly including upper vs. lower case letters.

Note

This API can only be used in AP mode.

Steps to enable EAP Authentication: Set the EAP Authentication mode and the EAP Server Parameters.

Command to set EAPAuthentication:

- call_qcsapi set_WPA_authentication_modes <device> EAP Authentication

Command to set Encryption:

- call_qcsapi set_WPA_encryption_modes $device <encryption>

Command to configure RADIUS authentication servers:

- call_qcsapi add_radius_auth_server_cfg <device> <ipaddr> <port> <sharedkey>
- call_qcsapi del_radius_auth_server_cfg <device> <ipaddr> <port>
- call_qcsapi get_radius_auth_server_cfg <device>

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
authentication_mode	a string containing a value per the **authentication type** table in the Security definitions section.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_WPA_authentication_modes <WiFi interface> <authentication
mode(s)>
```

Unless an error occurs, the output will be the string `complete`.

The authentication mode needs to be one of the values per the **authentication type** definitions table in the Security definitions section.

See also

qcsapi_SSID_set_authentication_modes

9.11.4.18 qcsapi_wifi_get_interworking()

```
int qcsapi_wifi_get_interworking (
            const char * ifname,
            string_32 interworking )
```

Get the 802.11u Interworking status.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
interworking	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_interworking <WiFi interface>
```

The output will be the interworking status unless an error occurs.

9.11.4.19 qcsapi_wifi_set_interworking()

```
int qcsapi_wifi_set_interworking (
            const char * ifname,
            const string_32 interworking )
```

Set the 802.11u Interworking status.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
interworking	0(Disable), 1(Enable)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi set_interworking <WiFi interface> <interworking>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.20 qcsapi_wifi_get_80211u_params()

```
int qcsapi_wifi_get_80211u_params (
            const char * ifname,
            const string_32 u_param,
            string_256 p_buffer )
```

Get the value for the specified 802.11u parameter, as specified by the qcsapi_80211u_params parameter, with the value returned in the address specified by p_buffer.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
u_param	is 802.11u parameter to get the value of
p_buffer	return parameter to contain the value of the 802.11u parameter

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_80211u_params <WiFi interface> <u_param>
```

The output will be the 802.11u parameter unless an error occurs.

9.11.4.21 qcsapi_wifi_set_80211u_params()

```
int qcsapi_wifi_set_80211u_params (
            const char * ifname,
            const string_32 param,
            const string_256 value1,
            const string_32 value2 )
```

Set the value for a specified 802.11u parameter.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
param	is the 802.11u parameter to set
value1	is the first value for the parameter
value2	is the second value for the parameter, or NULL if the parameter has only one value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

Valid parameters and their corresponding values shown in the following table. The exact format of the values for each parameter are described in the hostapd.conf man page.

Parameter	value1	value2
internet	0 or 1	-
access_network_type	type	-
network_auth_type	indicator value	-
hessid	MAC address	-
ipaddr_type_availability	IPv4 type	IPv6 type
domain_name	domain name	-
anqp_3gpp_cell_net	MCC1,MNC1;MCC2,MNC2;...	

Note

Max anqp_3gpp_cell_net count is IEEE80211U_3GPP_CELL_NET_MAX

`call_qcsapi` interface:

```
call_qcsapi set_80211u_params <WiFi interface> <param> <value1> <value2>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.22 qcsapi_security_get_nai_realms()

```
int qcsapi_security_get_nai_realms (
            const char * ifname,
            string_4096 p_value )
```

Get a list of the configured 802.11u NAI Realms.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	is pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

ifname must be the primary interface.

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_nai_realms <WiFi interface>
```

The output will be a list of NAI Realms unless an error occurs.

9.11.4.23 qcsapi_security_add_nai_realm()

```
int qcsapi_security_add_nai_realm (
            const char * ifname,
            const int encoding,
            const char * nai_realm,
            const char * eap_method )
```

Add or update an 802.11u NAI Realm.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
encoding	accepts value 0 or 1
nai_realm	
eap_method	

Note

If the NAI Realm already exists, it will be updated.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_nai_realm <WiFi interface> <encoding> <NAI realm> <EAP
methods>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.24 qcsapi_security_del_nai_realm()

```
int qcsapi_security_del_nai_realm (
            const char * ifname,
            const char * nai_realm )
```

Delete an existing 802.11u NAI Realm.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	nai_realm to be deleted

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_nai_realm <WiFi interface> <Nai realm>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.25 qcsapi_security_get_roaming_consortium()

```
int qcsapi_security_get_roaming_consortium (
            const char * ifname,
            string_1024 p_value )
```

Get the list of configured 802.11 Roaming Consortia.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	is pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_roaming_consortium <WiFi interface>
```

The output will be the roaming consortium value unless an error occurs.

9.11.4.26 qcsapi_security_add_roaming_consortium()

```
int qcsapi_security_add_roaming_consortium (
            const char * ifname,
            const char * p_value )
```

Add an 802.11u Roaming Consortium.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	the Roaming Consortium OI, which is a 3 to 15 octet hexadecimal string

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_roaming_consortium <WiFi interface> <OI>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.27 qcsapi_security_del_roaming_consortium()

```
int qcsapi_security_del_roaming_consortium (
            const char * ifname,
            const char * p_value )
```

Delete an existing 802.11 Roaming Consortium.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	roaming_consortium to be deleted

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_roaming_consortium <WiFi interface> <OI>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.28 qcsapi_security_get_venue_name()

```
int qcsapi_security_get_venue_name (
            const char * ifname,
            string_4096 p_value )
```

Get the list of configured 802.11 Venue names.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	is pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_venue_name <WiFi interface>
```

The output will be the list of venue names unless an error occurs.

9.11.4.29 qcsapi_security_add_venue_name()

```
int qcsapi_security_add_venue_name (
            const char * ifname,
            const char * lang_code,
            const char * venue_name )
```

Add an 802.11u venue name.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
lang_code	2 or 3 character ISO-639 language code. E.g. "eng" for English
name	venue name (Max IEEE80211U_VENUE_NAME_LEN_MAX characters)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_venue_name <WiFi interface> <lang_code> <name>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.30 qcsapi_security_del_venue_name()

```
int qcsapi_security_del_venue_name (
            const char * ifname,
            const char * lang_code,
            const char * venue_name )
```

Delete an 802.11u venue name.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
lang_code	2 or 3 character ISO-639 language code. E.g. "eng" for English
name	venue name (Max IEEE80211U_VENUE_NAME_LEN_MAX characters)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_venue_name <WiFi interface> <lang_code> <name>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.31 qcsapi_security_get_oper_friendly_name()

```
int qcsapi_security_get_oper_friendly_name (
            const char * ifname,
            string_4096 p_value )
```

Get the list of configured Hotspot 2.0 opererator friendly names.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	is pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_oper_friendly_name <WiFi interface>
```

The output will be the list of Hotspot 2.0 opererator friendly names unless an error occurs.

9.11.4.32 qcsapi_security_add_oper_friendly_name()

```
int qcsapi_security_add_oper_friendly_name (
            const char * ifname,
            const char * lang_code,
            const char * oper_friendly_name )
```

Add an Hotspot 2.0 opererator friendly name.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
lang_code	2 or 3 character ISO-639 language code. E.g. "eng" for English
name	Hotspot 2.0 opererator friendly name (Max HS20_OPER_FRIENDLY_NAME_LEN_MAX characters)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_oper_friendly_name <WiFi interface> <lang_code> <name>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.33 qcsapi_security_del_oper_friendly_name()

```
int qcsapi_security_del_oper_friendly_name (
            const char * ifname,
            const char * lang_code,
            const char * oper_friendly_name )
```

Delete an Hotspot 2.0 opererator friendly name.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
lang_code	2 or 3 character ISO-639 language code. E.g. "eng" for English
name	Hotspot 2.0 opererator friendly name (Max HS20_OPER_FRIENDLY_NAME_LEN_MAX characters)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_oper_friendly_name <WiFi interface> <lang_code> <name>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.34 qcsapi_security_get_hs20_conn_capab()

```
int qcsapi_security_get_hs20_conn_capab (
            const char * ifname,
            string_4096 p_value )
```

Get the list of configured Hotspot 2.0 connection capability.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_value	is pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_hs20_conn_capab <WiFi interface>
```

The output will be the list of Hotspot 2.0 connection capability unless an error occurs.

9.11.4.35 qcsapi_security_add_hs20_conn_capab()

```
int qcsapi_security_add_hs20_conn_capab (
            const char * ifname,
            const char * ip_proto,
            const char * port_num,
            const char * status )
```

Add an Hotspot 2.0 connection capability.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
ip_proto	is IP Protocol from 0 to IPPROTO_MAX
port_num	is Port Number from 0 to USHRT_MAX
status	is status of selected IP Protocol and Port Number. It can be 0 = Closed, 1 = Open, 2 = Unknown

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_hs20_conn_capab <WiFi interface> <ip_proto> <port_num>
<status>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.36 qcsapi_security_del_hs20_conn_capab()

```
int qcsapi_security_del_hs20_conn_capab (
            const char * ifname,
            const char * ip_proto,
            const char * port_num,
            const char * status )
```

Delete an Hotspot 2.0 connection capability.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
ip_proto	is IP Protocol from 0 to IPPROTO_MAX
port_num	is Port Number from 0 to USHRT_MAX
status	is status of selected IP Protocol and Port Number. It can be 0 = Closed, 1 = Open, 2 = Unknown.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_hs20_conn_capab <WiFi interface> <ip_proto> <port_num>
<status>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.37 qcsapi_wifi_get_hs20_status()

```
int qcsapi_wifi_get_hs20_status (
            const char * ifname,
            string_32 p_hs20 )
```

Get the Hotspot 2.0 parameter status.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_hs20	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_hs20_status <WiFi interface>
```

The output will be the hs status unless an error occurs.

9.11.4.38 qcsapi_wifi_set_hs20_status()

```
int qcsapi_wifi_set_hs20_status (
            const char * ifname,
            const string_32 hs20_val )
```

Enable or Disable Hotspot 2.0.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
hs20_val	either 0(Disable) or 1(Enable)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

If Hotspot 2.0 is enabled then WPS will be disabled.

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi set_hs20_status <WiFi interface> <hs>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.39 qcsapi_wifi_get_proxy_arp()

```
int qcsapi_wifi_get_proxy_arp (
            const char * ifname,
            string_32 p_proxy_arp )
```

Get the current Proxy ARP status.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_proxy_arp	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_proxy_arp <WiFi interface>
```

The output will be the Proxy ARP status unless an error occurs.

9.11.4.40 qcsapi_wifi_set_proxy_arp()

```
int qcsapi_wifi_set_proxy_arp (
            const char * ifname,
            const string_32 proxy_arp_val )
```

Set a Proxy ARP parameter.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
proxy_arp_val	0-Disable, 1-Enable

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi set_proxy_arp <WiFi interface> <proxy_arp>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.41 qcsapi_wifi_get_l2_ext_filter()

```
int qcsapi_wifi_get_l2_ext_filter (
            const char * ifname,
            const string_32 param,
            string_32 value )
```

Get the current L2 external filter parameters.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
param	parameter to get value of
value	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_l2_ext_filter <WiFi interface> <param>
```

The output will be the L2 external filter status unless an error occurs.

9.11.4.42 qcsapi_wifi_set_l2_ext_filter()

```
int qcsapi_wifi_set_l2_ext_filter (
            const char * ifname,
            const string_32 param,
            const string_32 value )
```

Set the L2 external filter parameters.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
param	parameter to set value of
value	value to be set for the parameter

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

ifname must be the primary interface.

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

Valid parameters and their corresponding values shown in the following table.

Parameter	Value
status	0 (Disable) or 1 (Enable)
port	emac0, emac1, pcie

`call_qcsapi` interface:

```
call_qcsapi set_l2_ext_filter <WiFi interface> <param> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.43 qcsapi_wifi_get_hs20_params()

```
int qcsapi_wifi_get_hs20_params (
            const char * ifname,
            const string_32 hs_param,
            string_32 p_buffer )
```

Get the value for the specified Hotspot 2.0 parameter. Refer to qcsapi_wifi_set_hs20_params for a list of valid parameter names.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
hs_param	Hotspot 2.0 parameter to get the value of
p_buffer	buffer pointer to contain the value of the requested parameter

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_hs20_params <WiFi interface> <hs_param>
```

The output will be the hs parameter unless an error occurs.

9.11.4.44 qcsapi_wifi_set_hs20_params()

```
int qcsapi_wifi_set_hs20_params (
            const char * ifname,
            const string_32 hs_param,
            const string_64 value1,
            const string_64 value2,
            const string_64 value3,
            const string_64 value4,
            const string_64 value5,
            const string_64 value6 )
```

Set a value for the specified Hotspot 2.0 parameter.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
hs_param	is hs parameter to set
value1-value6	values to be set for the parameter (value2 - value6 may be NULL)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

Valid parameters and their corresponding values are shown in the following table. The exact format of the values for each parameter are described in the hostapd.conf man page.

Parameter	value1	value2	value3	value4	value5	value6
hs20_wan_metrics	WAN Info	Downlink Speed	Uplink Speed	Downlink Load	Uplink Load	Load Measurement
disable_dgaf	0 or 1	-	-	-	-	-
hs20_operating_class	Single Band 2.4 GHz	Single Band 5 GHz	-	-	-	-
osu_ssid	SSID used for all OSU connections	-	-	-	-	-
osen	1 (enable) or 0 (disable)	-	-	-	-	-
hs20_deauth_req_timeout	Deauthentication request timeout in seconds	-	-	-	-	-

`call_qcsapi` interface:

```
call_qcsapi set_hs20_params <WiFi interface> <hs_param> <value1> <value2>
<value3> <value4> <value5> <value6>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.45 qcsapi_remove_11u_param()

```
int qcsapi_remove_11u_param (
            const char * ifname,
            const string_64 param )
```

Remove the value for the specified 802.11u parameter from hostapd.conf file, as specified by the qcsapi_11u_params parameter.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
param	802.11u parameter to be removed

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi remove_11u_param <WiFi interface> <param>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.46 qcsapi_remove_hs20_param()

```
int qcsapi_remove_hs20_param (
            const char * ifname,
            const string_64 hs_param )
```

Remove the specified Hotspot 2.0 parameter. Refer to qcsapi_wifi_set_hs20_params for a list of valid parameter names.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
hs_param	Hotspot 2.0 parameter to be removed

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi remove_hs20_param <WiFi interface> <hs_param>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.47 qcsapi_security_add_hs20_icon()

```
int qcsapi_security_add_hs20_icon (
            const char * ifname,
            const qcsapi_unsigned_int icon_width,
            const qcsapi_unsigned_int icon_height,
            const char * lang_code,
            const char * icon_type,
            const char * icon_name,
            const char * file_path )
```

Add a description for a Hotspot 2.0 icon.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
icon_width	icon width in pixels
icon_height	icon height in pixels
lang_code	2 or 3 character ISO-639 language code
icon_type	icon type e.g. "png"
icon_name	name for the icon
file_path	path to the image file; must be an existing file

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

There can be multiple icon descriptions, but the names must be unique.

`call_qcsapi` interface:

```
call_qcsapi add_hs20_icon <WiFi interface> <icon_width> <icon_height> <lang↩
_code> <icon_type> <icon_name> <file_path>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.48 qcsapi_security_get_hs20_icon()

```
int qcsapi_security_get_hs20_icon (
            const char * ifname,
            string_1024 value )
```

Get all configured Hotspot 2.0 icon descriptions. Each description has the following format.

```
<Icon Width>:<Icon Height>:<Language Code>:<Icon Type>:<Icon Name>:<File
Path>
```

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
value	pointer to buffer to contain returned icon descriptions

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_hs20_icon <WiFi interface>
```

The output will be the descriptions for Hotspot 2.0 icons, unless an error occurs.

9.11.4.49 qcsapi_security_del_hs20_icon()

```
int qcsapi_security_del_hs20_icon (
            const char * ifname,
            const string_1024 icon_name )
```

Delete the description for Hotspot 2.0 icon specified by name.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
icon_name	name of the icon to be deleted

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_hs20_icon <WiFi interface> <icon_name>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.50 qcsapi_security_add_osu_server_uri()

```
int qcsapi_security_add_osu_server_uri (
            const char * ifname,
            const string_256 osu_server_uri )
```

Add an Online Sign Up provider server URI. Each URI starts a new OSU provider description that might contain parameters added by qcsapi_security_add_osu_server_param.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
osu_server_uri	OSU provider URI

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

There can be multiple OSU provider servers, but each must have a unique URI.

`call_qcsapi` interface:

```
call_qcsapi add_osu_server_uri <WiFi interface> <osu_server_uri>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.51 qcsapi_security_get_osu_server_uri()

```
int qcsapi_security_get_osu_server_uri (
            const char * ifname,
            string_1024 value )
```

Get all configured Online Sign Up provider URIs. Each returned URI string is separated by a newline character.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
value	pointer to buffer to contain the returned strings

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_osu_server_uri <WiFi interface>
```

The output will be the URIs for all OSU providers, unless an error occurs.

9.11.4.52 qcsapi_security_del_osu_server_uri()

```
int qcsapi_security_del_osu_server_uri (
            const char * ifname,
            const string_256 osu_server_uri )
```

Delete an Online Sign Up provider server identified by a specified URI. All parameters for the specified provider are also deleted.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
osu_server_uri	OSU provider URI

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_osu_server_uri <WiFi interface> <osu_server_uri>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.53 qcsapi_security_add_osu_server_param()

```
int qcsapi_security_add_osu_server_param (
            const char * ifname,
            const string_256 osu_server_uri,
            const string_256 param,
            const string_256 value )
```

Add a parameter for an Online Sign Up provider identified by a specified URI. Valid parameters are described in the following table.

Parameter	Description	Multiple Entries
osu_friendly_name	Friendly name for OSU provider in the following format: `<Lang Code>:<Friendly Name>`	Yes
osu_nai	Network Access Identifier	No
osu_method_list	List of OSU methods separated by spaces. Valid values are: 0 - OMA DM, 1 - SOAP XML SPP	No
osu_icon	Name of the configured Hotspot 2.0 icon	Yes
osu_service_desc	OSU services description in the following format: `<Lang Code>:<Description>`	Yes

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
osu_server_uri	OSU provider URI
param	name of the parameter to be set
value	parameter value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi add_osu_server_param <WiFi interface> <osu_server_uri> <param>
<value>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.54 qcsapi_security_get_osu_server_param()

```
int qcsapi_security_get_osu_server_param (
            const char * ifname,
            const string_256 osu_server_uri,
            const string_256 param,
            string_1024 value )
```

Get all values for the specified parameter for an Online Sign Up provider. Value strings separated by new lines are placed in the returned buffer. Parameters are the same as for qcsapi_security_add_osu_server_param.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
osu_server_uri	URI of the OSU provider
param	parameter name
value	pointer to buffer to contain returned strings

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi get_osu_server_param <WiFi interface> <param>
```

The output will be the values for OSU provider parameter, unless an error occurs.

9.11.4.55 qcsapi_security_del_osu_server_param()

```
int qcsapi_security_del_osu_server_param (
            const char * ifname,
            const string_256 osu_server_uri,
            const string_256 param,
            const string_256 value )
```

Delete an Online Sign Up provider parameter with a specific value.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
osu_server_uri	OSU provider URI
param	parameter name
value	parameter value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

Refer to the WiFi Alliance Hotspot 2.0 (Release 2) Technical Specification for further information.

`call_qcsapi` interface:

```
call_qcsapi del_osu_server_param <WiFi interface> <osu_server_uri> <param>
<value>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.56 qcsapi_wifi_get_IEEE11i_encryption_modes()

```
int qcsapi_wifi_get_IEEE11i_encryption_modes (
            const char * ifname,
            string_32 encryption_modes )
```

See also

qcsapi_wifi_get_WPA_encryption_modes

9.11.4.57 qcsapi_wifi_set_IEEE11i_encryption_modes()

```
int qcsapi_wifi_set_IEEE11i_encryption_modes (
            const char * ifname,
            const string_32 encryption_modes )
```

See also

qcsapi_wifi_set_WPA_encryption_modes

9.11.4.58 qcsapi_wifi_get_IEEE11i_authentication_mode()

```
int qcsapi_wifi_get_IEEE11i_authentication_mode (
            const char * ifname,
            string_32 authentication_mode )
```

See also

qcsapi_wifi_get_WPA_authentication_mode

9.11.4.59 qcsapi_wifi_set_IEEE11i_authentication_mode()

```
int qcsapi_wifi_set_IEEE11i_authentication_mode (
            const char * ifname,
            const string_32 authentication_mode )
```

See also

qcsapi_wifi_set_WPA_authentication_mode

9.11.4.60 qcsapi_wifi_get_michael_errcnt()

```
int qcsapi_wifi_get_michael_errcnt (
            const char * ifname,
            uint32_t * errcount )
```

The total number of times the Michael integrity check has failed. This is an accumulated value of number of times MIC check failed starting from the beginning of device operation. Used for information purposes, it is not used directly for triggering Michael countermeasures event. Relevant only to WPA and 802.11i.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
errcount	a pointer to memory where MIC error count value should be placed

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_michael_errcnt <WiFi interface>
```

Unless an error occurs, the output will be the total number of Michael integrity check errors on specified interface.

9.11.4.61 qcsapi_wifi_get_pre_shared_key()

```
int qcsapi_wifi_get_pre_shared_key (
            const char * ifname,
            const qcsapi_unsigned_int key_index,
            string_64 pre_shared_key )
```

Get the WPA or RSN preshared key for an SSID.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
key_index	reserved - set to zero
pre_shared_key	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi get_pre_shared_key <WiFi interface> <key index>
call_qcsapi get_PSK <WiFi interface> <key index>
```

The output will be the preshared key unless an error occurs.

9.11.4.62 qcsapi_wifi_set_pre_shared_key()

```
int qcsapi_wifi_set_pre_shared_key (
            const char * ifname,
            const qcsapi_unsigned_int key_index,
            const string_64 pre_shared_key )
```

Set the WPA or RSN preshared key for an SSID.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
key_index	reserved - set to zero
pre_shared_key	a 64 hex digit PSK

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi set_pre_shared_key <WiFi interface> <key index> <preshared
key>

call_qcsapi set_PSK <WiFi interface> <key index> <preshared key>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.63 qcsapi_wifi_add_radius_auth_server_cfg()

```
int qcsapi_wifi_add_radius_auth_server_cfg (
            const char * ifname,
            const char * radius_auth_server_ipaddr,
            const char * radius_auth_server_port,
            const char * radius_auth_server_sh_key )
```

Add RADIUS authentication server configuration

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
radius_auth_server_ipaddr	- IP address of the RADIUS server
radius_auth_server_port	- Port of the RADIUS server
radius_auth_server_sh_key	- Shared secret key of the RADIUS server

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi add_radius_auth_server_cfg <WiFi interface> <ipaddr> <port>
<sh_key>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.64 qcsapi_wifi_del_radius_auth_server_cfg()

```
int qcsapi_wifi_del_radius_auth_server_cfg (
            const char * ifname,
            const char * radius_auth_server_ipaddr,
            const char * constp_radius_port )
```

Remove RADIUS authentication server configuration

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
radius_auth_server_ipaddr	- IP address of RADIUS server
radius_auth_server_port	- Port of the RADIUS server

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi del_radius_auth_server_cfg <WiFi interface> <ipaddr> <port>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.65 qcsapi_wifi_get_radius_auth_server_cfg()

```
int qcsapi_wifi_get_radius_auth_server_cfg (
            const char * ifname,
            string_1024 radius_auth_server_cfg )
```

Get RADIUS authentication servers configuration

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
radius_auth_server_cfg	- reads the RADIUS server configuraion

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi get_radius_auth_server_cfg <WiFi interface>
```

Unless an error occurs, the output will be the list of the RADIUS servers.

9.11.4.66 qcsapi_wifi_set_own_ip_addr()

```
int qcsapi_wifi_set_own_ip_addr (
            const char * ifname,
            const string_16 own_ip_addr )
```

Set the EAP own ip address of the AP.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
own_ip_addr	

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

This API can only be used in AP mode.

`call_qcsapi` interface:

```
call_qcsapi set_own_ip_addr <WiFi interface> <own ip addr>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.67 qcsapi_wifi_get_key_passphrase()

```
int qcsapi_wifi_get_key_passphrase (
            const char * ifname,
            const qcsapi_unsigned_int key_index,
            string_64 passphrase )
```

Returns the current WPA/11i passphrase. Applies to AP only. For a STA, use qcsapi_SSID_get_key_passphrase.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
key_index	- reserved, set to 0.
passphrase	a string to store the passphrase.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_key_passphrase <WiFi interface> 0
```

```
call_qcsapi get_passphrase <WiFi interface> 0
```

Unless an error occurs, the output will be the current passphrase.

The final '0' in the command line represents the key index.

See also

qcsapi_SSID_get_key_passphrase

9.11.4.68 qcsapi_wifi_set_key_passphrase()

```
int qcsapi_wifi_set_key_passphrase (
            const char * ifname,
            const qcsapi_unsigned_int key_index,
            const string_64 passphrase )
```

Sets the WPA/11i ASCII passphrase. Applies to AP only. For a STA, use qcsapi_SSID_set_key_passphrase.

By the WPA standard, the passphrase is required to have between 8 and 63 ASCII characters.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
key_index	- reserved, set to 0.
the	NULL terminated passphrase string, 8 - 63 ASCII characters (NULL termination not included in the count)

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_key_passphrase <WiFi interface> 0

call_qcsapi set_passphrase <WiFi interface> 0
```

Unless an error occurs, the output will be the string `complete`.

The final '0' in the command line represents the key index.

Note

The Linux shell processes the passphase parameter. Selected characters are interpreted by the shell, including the dollar sign ($), the backslash () and the backquote (`). We recommend putting the new passphrase in quotes and/or using the backslash character to "escape" characters that could be processed by the shell.

See also

qcsapi_SSID_get_key_passphrase

9.11.4.69 qcsapi_wifi_get_group_key_interval()

```
int qcsapi_wifi_get_group_key_interval (
            const char * ifname,
            unsigned int * p_key_interval )
```

Get the group key interval.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_key_interval	a pointer an integer to contain the group key rotation interval in seconds

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_group_key_interval <WiFi interface>
```

The output will be the group key interval unless an error occurs.

9.11.4.70 qcsapi_wifi_get_pairwise_key_interval()

```
int qcsapi_wifi_get_pairwise_key_interval (
            const char * ifname,
            unsigned int * p_key_interval )
```

Get the pairwise key rotation interval. This interval is used to timeout and cause a new PTK to be generated.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
p_key_interval	a pointer an integer to contain the pairwise key rotation interval in seconds.

Returns

0 if the command succeeded.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi get_pairwise_key_interval <WiFi interface>
```

The output will be the pairwise key interval unless an error occurs.

9.11.4.71 qcsapi_wifi_set_group_key_interval()

```
int qcsapi_wifi_set_group_key_interval (
            const char * ifname,
            const unsigned int key_interval )
```

Set the group key interval.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
key_interval	the group key rotation interval in seconds. Set to 0 for no key rotation.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_group_key_interval <WiFi interface> <group key interval>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.72 qcsapi_wifi_set_pairwise_key_interval()

```
int qcsapi_wifi_set_pairwise_key_interval (
            const char * ifname,
            const unsigned int key_interval )
```

Set the pairwise key rotation interval. This interval is used to timeout and cause a new PTK to be generated.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
key_interval	The pairwise key rotation interval in seconds. Set to 0 for no key rotation.

Returns

0 if the command succeeded.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi set_pairwise_key_interval <WiFi interface> <pairwise key interval>
```

Unless an error occurs, the output will be the string `complete`.

9.11.4.73 qcsapi_wifi_get_pmf()

```
int qcsapi_wifi_get_pmf (
            const char * ifname,
            int * p_pmf_cap )
```

Returns the current 802.11w pmf capability. Applies to AP.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
passphrase	an int rt.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_pmf<WiFi interface> 0

call_qcsapi get_pmf <WiFi interface> 0
```

Unless an error occurs, the output will be the current pmf capability.

See also

qcsapi_SSID_get_pmf

9.11.4.74 qcsapi_wifi_set_pmf()

```
int qcsapi_wifi_set_pmf (
            const char * ifname,
            int pmf_cap )
```

Sets the 802.11w / PMF capability. Applies to AP.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
pmf_cap.	

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_pmf <WiFi interface> 0
```

```
call_qcsapi set_pmf <WiFi interface> 0
```

Unless an error occurs, the output will be the string `complete`.

The final '0' in the command line represents the key index.

Note

The Linux shell processes the pmf parameter

See also

qcsapi_SSID_set_pmf

9.11.4.75 qcsapi_wifi_get_wpa_status()

```
int qcsapi_wifi_get_wpa_status (
            const char * ifname,
            char * wpa_status,
            const char * mac_addr,
            const qcsapi_unsigned_int max_len )
```

Returns the current WPA status. Only applies to AP.

Possible WPA status are: For AP

- `WPA_HANDSHAKING` - WPA handshaking started.
- `NO_WPA_HANDSHAKING` - WPA handshaking not started.
- `WPA_SUCCESS` - WPA handshaking is successful.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wpa_status	return parameter for storing the informative wpa_status string.
mac_addr	the mac_addr of the station that is connecting or connected to the AP.
max_len	the length of the wpa_status string passed in.

Returns

0 if the command succeeded.

a negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_wpa_status <WiFi interface> <Mac address>
```

Unless an error occurs, the output will be the current WPA handshaking status for the AP

9.11.4.76 qcsapi_wifi_get_psk_auth_failures()

```
int qcsapi_wifi_get_psk_auth_failures (
            const char * ifname,
            qcsapi_unsigned_int * count )
```

This API returns the total number of PSK authentication failures from the AP and associated stations.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
count	return parameter to contain the count of PSK authentication failures.

Returns

0 if the command succeeded.

a negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_psk_auth_failures <WiFi interface>
```

Unless an error occurs, the output will be the count of PSK authentication failures.

9.11.4.77 qcsapi_wifi_get_auth_state()

```
int qcsapi_wifi_get_auth_state (
            const char * ifname,
```

```
            const char * mac_addr,
            int * auth_state )
```

Returns the authenticated state(0/1). Only applies to AP.

Possible authenticated state are: For AP

- 1 - the station is authorized.
- 0 - the station is not authorized.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mac_addr	the mac_addr of the station.
auth_state	the state value to return .

Returns

0 if the command succeeded.

a negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_auth_state <WiFi interface> < Mac address>
```

Unless an error occurs, the output will be the authorized state for the station

9.11.4.78 qcsapi_wifi_set_security_defer_mode()

```
int qcsapi_wifi_set_security_defer_mode (
            const char * ifname,
            int defer )
```

This API call is used to set the current hostapd/wpa_supplicant configuration mode for a given interface

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
defer	will indicate the current hostapd/wpa_supplicant configuration mode 0: immediate mode 1:defer mode

Note

This API works only for wifi0 interface.

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

call_qcsapi set_security_defer_mode <wifi interface> {0 | 1}

9.11.4.79 qcsapi_wifi_get_security_defer_mode()

```
int qcsapi_wifi_get_security_defer_mode (
            const char * ifname,
            int * defer )
```

This API call is used to get the current hostapd/wpa_supplicant configuration mode for a given interface

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
defer	will store the current hostapd/wpa_supplicant configuration mode 0: immediate mode 1:defer mode

Note

This API works only for wifi0 interface.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi get_security_defer_mode <wifi interface>
```

9.11.4.80 qcsapi_wifi_apply_security_config()

```
int qcsapi_wifi_apply_security_config (
            const char * ifname )
```

This API call is used to configure/reconfigure the current hostapd/wpa_supplicant configration.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Note

This API works across all wifi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi apply_security_config <wifi interface>
```

9.12 MAC Address Filtering APIs

The AP can block a selected station from associating based on its MAC (hardware interface) address. This section lists and describes the APIs that manage this capability. The interface parameter to these APIs must reference a Wireless Extension (WE) device configured as an AP.
By default, MAC address filtering is disabled. Use the Set MAC Address Filtering API to configure this capability. Consult section 5.1.1 on the format of a MAC address when working with the scripting interface to the MAC address filtering APIs.

Functions

- int qcsapi_wifi_set_mac_address_filtering (const char *ifname, const qcsapi_mac_address_filtering new_↩ mac_address_filtering)

  Set MAC address filtering for the given interface.
- int qcsapi_wifi_get_mac_address_filtering (const char *ifname, qcsapi_mac_address_filtering *current_↩ mac_address_filtering)

  Get MAC Address Filtering.
- int qcsapi_wifi_authorize_mac_address (const char *ifname, const qcsapi_mac_addr address_to_authorize)

  Authorize a MAC address for MAC address filtering.
- int qcsapi_wifi_authorize_mac_address_list (const char *ifname, const int num, const qcsapi_mac_addr_list address_list_to_authorize)

  Authorize set of MAC addresses for MAC address filtering.
- int qcsapi_wifi_deny_mac_address (const char *ifname, const qcsapi_mac_addr address_to_deny)

  Block MAC addresses using the MAC address filtering feature.
- int qcsapi_wifi_deny_mac_address_list (const char *ifname, const int num, const qcsapi_mac_addr_list address_list_to_deny)

  Block MAC addresses using the MAC address filtering feature.
- int qcsapi_wifi_remove_mac_address (const char *ifname, const qcsapi_mac_addr address_to_remove)

  Remove MAC address from the MAC address filtering list.
- int qcsapi_wifi_remove_mac_address_list (const char *ifname, const int num, const qcsapi_mac_addr_list address_list_to_remove)

  Remove MAC address from the MAC address filtering list.
- int qcsapi_wifi_is_mac_address_authorized (const char *ifname, const qcsapi_mac_addr address_to_verify, int *p_mac_address_authorized)

  Check whether a MAC address is authorized.
- int qcsapi_wifi_get_authorized_mac_addresses (const char *ifname, char *list_mac_addresses, const unsigned int sizeof_list)

  Get a list of authorized MAC addresses.
- int qcsapi_wifi_get_denied_mac_addresses (const char *ifname, char *list_mac_addresses, const unsigned int sizeof_list)

  Get a list of denied MAC addresses.
- int qcsapi_wifi_set_accept_oui_filter (const char *ifname, const qcsapi_mac_addr oui, int flag)

  API to set OUI to filter list.
- int qcsapi_wifi_get_accept_oui_filter (const char *ifname, char *oui_list, const unsigned int sizeof_list)

  API to get of OUI filter list.
- int qcsapi_wifi_clear_mac_address_filters (const char *ifname)

  Clear the MAC address lists.
- int qcsapi_wifi_authorize_mac_address_list_ext (const char *ifname, struct qcsapi_mac_list *auth_mac_list)

  Authorize set of MAC addresses for MAC address filtering.
- int qcsapi_wifi_deny_mac_address_list_ext (const char *ifname, struct qcsapi_mac_list *deny_mac_list)

  Block MAC addresses using the MAC address filtering feature.
- int qcsapi_wifi_remove_mac_address_list_ext (const char *ifname, struct qcsapi_mac_list *remove_mac_↩ list)

  Remove MAC addresses from the MAC address filtering list.

9.12.1 Detailed Description

Note

All MAC Address Filtering APIs work with security configuration files. Their location is determined by the get file path configuration API (section File Path configuration). Results from these APIs may be inconsistent or incorrect if the file path to this security configuration files has not been correctly configured.

9.12.2 Data Type to Configure MAC Address Filtering

9.12.3 Error Codes from MAC Address Filtering APIs

The API that returns a list of authorized MAC addresses will fail with an error code of Configuration Error if the MAC address filtering is not set to Deny Unless Authorized. The API that returns a list of denied or blocked MAC addresses will fail with error code Configuration Error if the MAC address filtering is not set to Authorize Unless Denied.
Both of those APIs will fail with an error code of Buffer Overflow if the length of the string is too short to store all MAC addresses.
See QCSAPI Return Values for more details on error codes and error messages.

9.12.4 Function Documentation

9.12.4.1 qcsapi_wifi_set_mac_address_filtering()

```
int qcsapi_wifi_set_mac_address_filtering (
            const char * ifname,
            const qcsapi_mac_address_filtering new_mac_address_filtering )
```

Set the current MAC address filtering, based on the input parameters

If the MAC address filtering was configured as Disabled (`qcsapi_disable_mac_address_filtering`), calling the API to deny access to a MAC address will change the configuration to Accept unless Denied (`qcsapi_accept_mac_address_unless_denied`), so the referenced MAC address will be blocked from associating.

Note that MAC address filtering is disabled by default.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_mac_address_filtering	the new MAC address filtering mode to enable.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_macaddr_filter <WiFi interface> <0|1|2>
```

Final argument configures the MAC address filtering:

- 0 to disable MAC address filtering,
- 1 to accept an association unless the MAC address has been blocked,
- 2 to block associations unless the MAC address has been authorized.

These values match those in the enumerated data type qcsapi_mac_address_filtering. Unless an error occurs, the output will be the string `complete`.

Note

If the MAC address filtering is set to Accept Unless Blocked, and MAC address filtering is turned off, the list of blocked MAC addresses will be lost.

See also

qcsapi_mac_address_filtering

9.12.4.2 qcsapi_wifi_get_mac_address_filtering()

```
int qcsapi_wifi_get_mac_address_filtering (
            const char * ifname,
            qcsapi_mac_address_filtering * current_mac_address_filtering )
```

Get the current MAC address filtering. Returned value will matches one of those in the enumerated data type.

This is the dual function of qcsapi_wifi_set_mac_address_filtering.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_mac_address_filtering	return parameter to contain the MAC address filtering mode currently enabled.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_macaddr_filter <WiFi interface>
```

Unless an error occurs, the output will be 0, 1 or 2, representing the enumerators in the enumeration qcsapi_mac↩_address_filtering.

9.12.4.3 qcsapi_wifi_authorize_mac_address()

```
int qcsapi_wifi_authorize_mac_address (
            const char * ifname,
            const qcsapi_mac_addr address_to_authorize )
```

Authorize the referenced MAC address against the MAC address filtering function.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
address_to_authorize	the MAC address of the device to authorize.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi authorize_macaddr <WiFi interface> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

See Format for a MAC address for details on the format for entering the MAC address.

9.12.4.4 qcsapi_wifi_authorize_mac_address_list()

```
int qcsapi_wifi_authorize_mac_address_list (
            const char * ifname,
            const int num,
            const qcsapi_mac_addr_list address_list_to_authorize )
```

Authorize the referenced MAC addresses (up to 8) against the MAC address filtering function.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
num	number of MAC address
address_list_to_authorize	the MAC addresses of the device to authorize.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi authorize_macaddr <WiFi interface> <MAC address0> [MAC address1]
...  [MAC address7]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

Note

This API is deprecated and replaced with qcsapi_wifi_authorize_mac_address_list_ext.

9.12.4.5 qcsapi_wifi_deny_mac_address()

```
int qcsapi_wifi_deny_mac_address (
            const char * ifname,
            const qcsapi_mac_addr address_to_deny )
```

Block the referenced MAC address. If the MAC address filtering was configured as disabled, calling this API will change the configuration to accept unless denied.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
address_to_deny	the MAC address to deny.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi deny_macaddr <WiFi interface> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

See Format for a MAC address for details on the format for entering the MAC address.

9.12.4.6 qcsapi_wifi_deny_mac_address_list()

```
int qcsapi_wifi_deny_mac_address_list (
            const char * ifname,
            const int num,
            const qcsapi_mac_addr_list address_list_to_deny )
```

Block the referenced MAC addresses (up to 8). If the MAC address filtering was configured as disabled, calling this API will change the configuration to accept unless denied.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
num	number of MAC address
address_list_to_deny	the MAC addresses to deny.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi deny_macaddr <WiFi interface> <MAC address> [MAC address1] ...
[MAC address7]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

Note

This API is deprecated and replaced with qcsapi_wifi_deny_mac_address_list_ext.

9.12.4.7 qcsapi_wifi_remove_mac_address()

```
int qcsapi_wifi_remove_mac_address (
            const char * ifname,
            const qcsapi_mac_addr address_to_remove )
```

Remove the referenced MAC address.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
address_to_remove	the MAC address to remove.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi remove_macaddr <WiFi interface> <MAC address>
```

Unless an error occurs, the output will be the string `complete`.

See Format for a MAC address for details on the format for entering the MAC address.

9.12.4.8 qcsapi_wifi_remove_mac_address_list()

```
int qcsapi_wifi_remove_mac_address_list (
            const char * ifname,
            const int num,
            const qcsapi_mac_addr_list address_list_to_remove )
```

Remove the referenced MAC addresses (up to 8).

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
num	number of MAC address
address_list_to_remove	the MAC address to remove.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi remove_macaddr <WiFi interface> <MAC address> [MAC address1]
...  [MAC address7]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

Note

This API is deprecated and replaced with qcsapi_wifi_remove_mac_address_list_ext.

9.12.4.9 qcsapi_wifi_is_mac_address_authorized()

```
int qcsapi_wifi_is_mac_address_authorized (
            const char * ifname,
            const qcsapi_mac_addr address_to_verify,
            int * p_mac_address_authorized )
```

Reports whether a STA with the referenced MAC address is authorized, that is, MAC address filtering will allow the STA to associate.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
address_to_verify	the MAC address to check for authorization.
p_mac_address_authorized	return parameter to indicate authorized (1) or not authorized (0).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi is_macaddr_authorized <WiFi interface> <MAC address>
```

See Format for a MAC address for details on the format for entering the MAC address.

Unless an error occurs, the output is either 1 (MAC address can associate) or 0 (MAC address will be blocked from associating).

9.12.4.10 qcsapi_wifi_get_authorized_mac_addresses()

```
int qcsapi_wifi_get_authorized_mac_addresses (
            const char * ifname,
            char * list_mac_addresses,
            const unsigned int sizeof_list )
```

Get a list of authorized MAC addresses. MAC address filtering must have been configured to Deny unless Authorized (`qcsapi_deny_mac_address_unless_authorized`).

MAC addresses will be returned in `list_mac_addresses` up to the size of the parameter, as expressed in `sizeof_list`, in the standard format for MAC addresses, separated by commas.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
list_mac_addresses	return parameter to contain the list of comma delimited MAC addresses
sizeof_list	the size of the input list_mac_addresses buffer.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_authorized_macaddr <WiFi interface> <size of string>
```

Unless an error occurs, output will be the list of authorized MAC addresses, separated by commas. MAC address filtering must be set to deny unless authorized (2); use `call_qcsapi get_macaddr_filter` to verify this.

Final parameter is the size of the string to receive the list of authorized MAC addresses.

9.12.4.11 qcsapi_wifi_get_denied_mac_addresses()

```
int qcsapi_wifi_get_denied_mac_addresses (
            const char * ifname,
            char * list_mac_addresses,
            const unsigned int sizeof_list )
```

Get a list of denied or blocked MAC addresses. MAC address filtering must have been configured to accept unless denied (`qcsapi_accept_mac_address_unless_denied`). MAC addresses will be returned in `list_mac_addresses` up to the size of the passed in buffer, as expressed in `sizeof_list`, in the standard format for MAC addresses, separated by commas.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
list_mac_addresses	return parameter to contain the list of comma delimited MAC addresses
sizeof_list	the size of the input list_mac_addresses buffer.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_denied_macaddr <WiFi interface> <size of string>
```

Unless an error occurs, output will be the list of denied MAC addresses, separated by commas. MAC address filtering must be set to accept unless denied (1); use `call_qcsapi get_macaddr_filter` to verify this. Final parameter is the size of the string to receive the list of denied MAC addresses.

9.12.4.12 qcsapi_wifi_set_accept_oui_filter()

```
int qcsapi_wifi_set_accept_oui_filter (
            const char * ifname,
            const qcsapi_mac_addr oui,
            int flag )
```

This function can be called to set OUI into filter list.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
oui	Organizationally unique identifier string in full MAC address format.
flag	1 to insert OUI and 0 to remove OUI to/from white list

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values. for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_accept_oui_filter <WiFi interface> <OUI> <1 | 0>
```

Unless an error occurs, the output will be "complete".

9.12.4.13 qcsapi_wifi_get_accept_oui_filter()

```
int qcsapi_wifi_get_accept_oui_filter (
            const char * ifname,
            char * oui_list,
            const unsigned int sizeof_list )
```

This function can be called to get OUI filter list.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
oui_list	Where to receive oui in string format.
sizeof_list	Specifies the size of string that prepare for the list return.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_accept_oui_filter <WiFi interface> [size]
```

Unless an error occurs, the output will be string that contains MAC address separated by comma.

9.12.4.14 qcsapi_wifi_clear_mac_address_filters()

```
int qcsapi_wifi_clear_mac_address_filters (
            const char * ifname )
```

This function can be called to clear any accept or deny lists created using the MAC address filtering APIs.

After this is called, the hostapd.deny and hostapd.accept files will be reset to default - any existing MAC addresses in these files will be cleared.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi clear_mac_filters <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.12.4.15 qcsapi_wifi_authorize_mac_address_list_ext()

```
int qcsapi_wifi_authorize_mac_address_list_ext (
            const char * ifname,
            struct qcsapi_mac_list * auth_mac_list )
```

Authorize the referenced MAC addresses (up to 200) against the MAC address filtering function.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
auth_mac_list	the list and the count of device MAC addresses to authorize.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi authorize_macaddr <WiFi interface> <MAC address0> [MAC address1]
...  [MAC address199]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

9.12.4.16 qcsapi_wifi_deny_mac_address_list_ext()

```
int qcsapi_wifi_deny_mac_address_list_ext (
            const char * ifname,
            struct qcsapi_mac_list * deny_mac_list )
```

Block the referenced MAC addresses (up to 200). If the MAC address filtering was configured as disabled, calling this API will change the configuration to accept unless denied.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
deny_mac_list	the list and the count of MAC addresses to deny.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi deny_macaddr <WiFi interface> <MAC address> [MAC address1] ...
[MAC address199]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

9.12.4.17 qcsapi_wifi_remove_mac_address_list_ext()

```
int qcsapi_wifi_remove_mac_address_list_ext (
            const char * ifname,
            struct qcsapi_mac_list * remove_mac_list )
```

Remove the referenced MAC addresses (up to 200).

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
remove_mac_list	the list and the count of MAC addresses to remove.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi remove_macaddr <WiFi interface> <MAC address> [MAC address1]
...  [MAC address199]
```

Unless an error occurs, the output will be the string `complete`.

See section here for details on the format for entering the MAC address.

See section here for more details on error codes and error messages.

9.13 MAC Address Reservation APIs

MAC address reservation can be used to prevent associated WiFi devices and downstream devices from hijacking MAC addresses that belong to core network devices.

Functions

- int qcsapi_wifi_set_mac_address_reserve (const char *ifname, const char *addr, const char *mask)

  Set MAC address reservation.
- int qcsapi_wifi_get_mac_address_reserve (const char *ifname, string_256 buf)

  Get MAC address reservation.
- int qcsapi_wifi_clear_mac_address_reserve (const char *ifname)

  Clear MAC address reservation.

9.13.1 Detailed Description

MAC address reservation is implemented on an Access Point by configuring a list of up to six reserved MAC addresses. An optional mask can be supplied with each entry in order to reserve a range of MAC addresses.

The following example reserves 1c:6f:65:d1:bf:01 and the 16 MAC addresses from 1c:6f:65:d1:bf:10 to 1c:6f:65:d1:bf:1f are reserved for devices on the wired side of the Access Point.

```
call_qcsapi set_macaddr_reserve wifi0 1c:6f:65:d1:bf:01

call_qcsapi set_macaddr_reserve wifi0 1c:6f:65:d1:bf:10 ff:ff:ff:ff:ff:f0
```

Any association request with a source address that matches an entry in the reserved MAC address list is refused, and any Ethernet packet with a source address that matches an entry in the reserved MAC address list is dropped.

9.13.2 Function Documentation

9.13.2.1 qcsapi_wifi_set_mac_address_reserve()

```
int qcsapi_wifi_set_mac_address_reserve (
            const char * ifname,
            const char * addr,
            const char * mask )
```

Prevent selected MAC addresses from being used by WiFi stations or back-end devices.

This feature can be used to ensure that MAC addresses of core networking devices cannot be hijacked by WiFi stations or by devices connected to WiFi stations.

Note

This API can only be used in AP mode.

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
addr	MAC address to be reserved
mask	MAC address mask in the same format as a MAC address, or an empty string for a single MAC address

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

See Format for a MAC address for details on the format for entering the MAC address.

`call_qcsapi` interface:

```
call_qcsapi set_macaddr_reserve <WiFi interface> <addr> [<mask>]
```

Unless an error occurs, the output will be the string `complete`.

Note

A maximum of 6 MAC addresses and/or MAC address ranges may be reserved.

9.13.2.2 qcsapi_wifi_get_mac_address_reserve()

```
int qcsapi_wifi_get_mac_address_reserve (
            const char * ifname,
            string_256 buf )
```

Get the list of reserved MAC addresses.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
buf	pointer to a buffer for storing the returned list

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_macaddr_reserve <WiFi interface>
```

Unless an error occurs, the output will be a list of reserved MAC addresses and masks.

9.13.2.3 qcsapi_wifi_clear_mac_address_reserve()

```
int qcsapi_wifi_clear_mac_address_reserve (
            const char * ifname )
```

Delete all MAC address reservation configuration.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi clear_macaddr_reserve <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.14 Options

A variety of options can be accessed or set. See the discussion of the datatype qcsapi_option_type for the list of available options. See the discussion of the datatype qcsapi_wifi_param_type for the list of available parameters. Relevant entry points follow:

Functions

- int qcsapi_wifi_get_option (const char *ifname, qcsapi_option_type qcsapi_option, int *p_current_option)

  Get a WiFi option.
- int qcsapi_wifi_set_option (const char *ifname, qcsapi_option_type qcsapi_option, int new_option)

  Set a WiFi option.
- int qcsapi_get_board_parameter (qcsapi_board_parameter_type board_param, string_64 p_buffer)

  Get a board related parameter.
- int qcsapi_get_swfeat_list (string_4096 p_buffer)

  Get the feature list.
- int qcsapi_wifi_get_parameter (const char *ifname, qcsapi_wifi_param_type type, int *p_value)

  Get a WiFi parameter.
- int qcsapi_wifi_set_parameter (const char *ifname, qcsapi_wifi_param_type type, int value)

  Set a WiFi parameter.

9.14.1 Detailed Description

9.14.2 WiFi Options and the call_qcsapi Interface

The table below lists selected options as listed in the enumerated type and how to pass them to call_qcsapi.

Option	call_qcsapi representation
qcsapi_channel_refresh	channel_refresh
qcsapi_DFS	DFS
qcsapi_wmm	WMM
qcsapi_beacon_advertise	beacon_advertise
qcsapi_wifi_radio	radio
qcsapi_autorate_fallback	autorate
qcsapi_security	security
qcsapi_SSID_broadcast	SSID_broadcast
qcsapi_short_GI	shortGI
qcsapi_802_11h	802_11h
qcsapi_tpc_query	tpc_query
qcsapi_dfs_fast_channel_switch	dfs_fast_switch
qcsapi_dfs_avoid_dfs_scan	avoid_dfs_scan
qcsapi_uapsd	uapsd
qcsapi_sta_dfs	sta_dfs
qcsapi_specific_scan	specific_scan
qcsapi_GI_probing	GI_probing
qcsapi_GI_fixed	GI_fixed
qcsapi_stbc	stbc
qcsapi_beamforming	beamforming

qcsapi_short_slot	short_slot
qcsapi_short_preamble	short_preamble
qcsapi_rts_cts	rts_cts
qcsapi_40M_only	40M_bw_only
qcsapi_obss_coexist	obss_coexist
qcsapi_11g_protection	11g_protection
qcsapi_11n_protection	11n_protection

To access the get option API enter:
`call_qcsapi get_option wifi0 <option>`
Unless an error occurs, the output will be either TRUE or FALSE.
To access the set option API enter:
`call_qcsapi set_option wifi0 <option> <1 | TRUE | 0 | FALSE>`
Unless an error occurs, the output will be the string `complete`.

9.14.3 Notes on Selected Options

- The autorate fallback option (qcsapi_autorate_fallback) can be considered a rate setting. This option can be enabled from the set option API, but disabling is not allowed. To disable autorate fallback, call the Set MCS rate API with a valid MCS rate.
- WiFi MultiMedia (WMM, qcsapi_wmm) is required for 802.11n. As Quantenna devices always operate in 802.11n mode, this option is enabled by default and thus cannot be disabled thru the Set Option API.
- Dynamic Frequency Selection (DFS, qcsapi_DFS) is a read-only option. If enabled, the programming on the Quantenna WiFi device supports DFS. It is not possible to enable or disable DFS through the set option API.
- SSID Broadcast controls whether the name of the SSID is included in beacons broadcast by the AP. This option is not available if the device is configured as a STA.
- Security is a read-only option. On an AP, security is determined by the Set Beacon API. On a STA, security is determined by the security policy of the AP it associates with.
- DFS Fast Switch enhances availability if a channel covered by DFS / radar protocols is selected. The protocol requires the AP to immediately switch channels if radar is detected on the current channel. By default, the AP scans available channels to find the channel with least interference. This operation typically leads to a gap in traffic lasting from 20 seconds to over 1 minute. With DFS Fast Switch enabled, the AP immediately switches to a non-DFS channel. Testing with this option enabled shows there should be no loss of traffic if radar is detected.
  If both DFS Fast Switch and Avoid DFS Scan are enabled, DFS Fast Switch takes precedence.
  Examples using `call_qcsapi`:
  To enable DFS Fast Swich:
  `call_qcsapi set_option wifi0 dfs_fast_switch 1`
  To disable DFS Fast Switch:
  `call_qcsapi set_option wifi0 dfs_fast_switch 0`
  To query this option:
  `call_qcsapi get_option wifi0 dfs_fast_switch`

- Avoid DFS Scan causes the AP to scan only non-DFS channels if radar is detected and a switch of channels is required. Enabling this option ensures that the Channel Availability Check (CAC) will not be required after radar is detected. A gap in traffic should still be expected after radar is detected, but without the CAC, the maximum gap should be less than 30 seconds.
  If both DFS Fast Switch and Avoid DFS Scan are enabled, DFS Fast Switch takes precedence.
  Examples using call_qcsapi:
  To enable Avoid DFS Fast Swich:

`call_qcsapi set_option wifi0 avoid_dfs_scan 1`
To disable DFS Fast Switch:
`call_qcsapi set_option wifi0 avoid_dfs_scan 0`
To query this option:
`call_qcsapi get_option wifi0 avoid_dfs_scan`

- For options that are not supported yet, API will return `-qcsapi_option_not_supported` and for call_qcsapi, it shows
`QCS API error 1044: Option is not supported`
Caller can check the return value `-qcsapi_option_not_supported` to query if the option is supported.

9.14.4 WiFi Parameters and the call_qcsapi Interface

The table below lists selected parameters as listed in the enumerated type and how to pass them to call_qcsapi.

Option	call_qcsapi representation
qcsapi_wifi_param_dtim_period	dtim_period

To access the get parameter API enter:
`call_qcsapi get_wifi_param wifi0 <parameter>`
Unless an error occurs, the output will be the numeric value of the parameter.
To access the set parameter API enter:
`call_qcsapi set_wifi_param wifi0 <parameter> <numeric value>`
Unless an error occurs, the output will be the string `complete`.

9.14.5 Notes on Selected Parameters

- The dtim period is an AP only parameter.
It controls how often the broadcast/multicast traffic indication bit set in beacon unit.

9.14.6 Function Documentation

9.14.6.1 qcsapi_wifi_get_option()

```
int qcsapi_wifi_get_option (
            const char * ifname,
            qcsapi_option_type qcsapi_option,
            int * p_current_option )
```

Get the value for an option, as specified by the qcsapi_option_type parameter, with the value returned in the address specified by p_current_option.

Note

Value will be either 0 (disabled, false) or 1 (enabled, true). If the option (feature) is not supported, the API will return `-qcsapi_option_not_supported`.

Two options short_gi and stbc are global and can only be configured on primary interface wifi0.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
qcsapi_option	the option to get the value of.
p_current_option	return parameter to contain the value of the option.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_option <WiFi interface> <option>
```

See also

qcsapi_option_type

9.14.6.2 qcsapi_wifi_set_option()

```
int qcsapi_wifi_set_option (
            const char * ifname,
            qcsapi_option_type qcsapi_option,
            int new_option )
```

Set the value for an option, as specified by the qcsapi_option_type parameter, with a value of 0 disabling the option or setting it to false and a value of 1 enabling the option or setting it to true. A non-zero value will be interpreted as 1.

Note

Not all options can be set. If the feature is not supported, the API returns `-qcsapi_option_not_supported`. For some options having fixed value, the API will return `-EOPNOTSUPP`.

Two options short_gi and stbc are global and can only be configured on primary interface wifi0.

Enabling qlink option is equivalent to setting it to auto mode (network iface autoselection). Use qcsapi_config_update_parameter to set qlink option to a specific network interface name.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
qcsapi_option	the option to set the value of.
new_option	the new option value..

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_option <WiFi interface> <option> <1 | TRUE | 0 | FALSE>
```

See also

qcsapi_option_type. For the qcsapi_option_type 'sta_dfs', after enable/disable sta_dfs, you must also update the power table using the command call_qcsapi restore_regulatory_tx_power wifi0

9.14.6.3 qcsapi_get_board_parameter()

```
int qcsapi_get_board_parameter (
            qcsapi_board_parameter_type board_param,
            string_64 p_buffer )
```

Get the value for the specified board parameter, as specified by the qcsapi_board_parameter_type parameter, with the value returned in the address specified by p_buffer.

If the parameter (feature) is not supported, the API will return `-qcsapi_board_parameter_not_supported`.

Parameters

board_param	the board parameter to get the value of.
p_buffer	return parameter to contain the value of the board parameter.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_board_parameter <board parameter>
```

See also

qcsapi_board_parameter_type

9.14.6.4 qcsapi_get_swfeat_list()

```
int qcsapi_get_swfeat_list (
             string_4096 p_buffer )
```

Get a list of features supported on this device.

Parameters

p_buffer	buffer to return the supported features list

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_swfeat_list
```

9.14.6.5 qcsapi_wifi_get_parameter()

```
int qcsapi_wifi_get_parameter (
            const char * ifname,
            qcsapi_wifi_param_type type,
            int * p_value )
```

Get the value for a parameter with the value returned in the address specified by p_value. Parameters supported can be showed by QCSAPI help option.
`call_qcsapi -h wifi_parameters`

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
parameter_name	the parameter to get the value of.
p_value	return the value of the parameter.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values. for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_wifi_parameter <WiFi interface> <parameter>
```

See also

qcsapi_wifi_param_type

9.14.6.6 qcsapi_wifi_set_parameter()

```
int qcsapi_wifi_set_parameter (
            const char * ifname,
            qcsapi_wifi_param_type type,
            int value )
```

Set the value for an parameter. Parameters supported can be showed by QCSAPI help option.
`call_qcsapi -h wifi_parameters`

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
parameter_name	the parameter to set the value of.
value	the value to be set for the parameter.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values. for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_wifi_parameter <WiFi interface> <parameter> <value>
```

See also

qcsapi_wifi_param_type

9.15 SSID APIs

The WPA Supplicant configuration file (STA only) allows multiple service sets to be configured.

Functions

- int qcsapi_SSID_create_SSID (const char *ifname, const qcsapi_SSID new_SSID)

  Create a new SSID.
- int qcsapi_SSID_remove_SSID (const char *ifname, const qcsapi_SSID del_SSID)

  Remove an existing SSID.
- int qcsapi_SSID_verify_SSID (const char *ifname, const qcsapi_SSID current_SSID)

  Verify an SSID is present.
- int qcsapi_SSID_rename_SSID (const char *ifname, const qcsapi_SSID current_SSID, const qcsapi_SSID new_SSID)

  Rename an existing SSID.
- int qcsapi_SSID_get_SSID_list (const char *ifname, const unsigned int arrayc, char **list_SSID)

  Get the list of SSIDs.
- int qcsapi_SSID_set_protocol (const char *ifname, const qcsapi_SSID current_SSID, const char *new_↩ protocol)

  Set the authentication protocol for an SSID.
- int qcsapi_SSID_get_protocol (const char *ifname, const qcsapi_SSID current_SSID, string_16 current_↩ protocol)

  Get the security protocol for an SSID.
- int qcsapi_SSID_get_encryption_modes (const char *ifname, const qcsapi_SSID current_SSID, string_32 encryption_modes)

  Get the encryption modes for an SSID.
- int qcsapi_SSID_set_encryption_modes (const char *ifname, const qcsapi_SSID current_SSID, const string_32 encryption_modes)

  Set the encryption mode for an SSID.
- int qcsapi_SSID_get_group_encryption (const char *ifname, const qcsapi_SSID current_SSID, string_32 encryption_mode)

  Get the group encryption cipher.
- int qcsapi_SSID_set_group_encryption (const char *ifname, const qcsapi_SSID current_SSID, const string_32 encryption_mode)

  Set the group encryption cipher.
- int qcsapi_SSID_get_authentication_mode (const char *ifname, const qcsapi_SSID current_SSID, string_32 authentication_mode)

  Get the authentication mode for an SSID.
- int qcsapi_SSID_set_authentication_mode (const char *ifname, const qcsapi_SSID current_SSID, const string_32 authentication_mode)

  Set the authentication mode for an SSID.
- int qcsapi_SSID_get_pre_shared_key (const char *ifname, const qcsapi_SSID current_SSID, const qcsapi_unsigned_int key_index, string_64 pre_shared_key)

  Get the preshared key.
- int qcsapi_SSID_set_pre_shared_key (const char *ifname, const qcsapi_SSID current_SSID, const qcsapi_unsigned_int key_index, const string_64 pre_shared_key)

  Set the preshared key.
- int qcsapi_SSID_get_key_passphrase (const char *ifname, const qcsapi_SSID current_SSID, const qcsapi_unsigned_int key_index, string_64 passphrase)

  Get the passphrase for an SSID.

- int qcsapi_SSID_set_key_passphrase (const char *ifname, const qcsapi_SSID current_SSID, const qcsapi_unsigned_int key_index, const string_64 passphrase)

  Set the passphrase (ASCII) for an SSID.
- int qcsapi_wifi_update_bss_cfg (const char *ifname, const qcsapi_wifi_mode wifi_mode, const char *ssid, const char *param_name, const char *param_value, const char *param_type)

  Manipulate a specific paramter of security deamon.
- int qcsapi_SSID_get_pmf (const char *ifname, const qcsapi_SSID current_SSID, int *p_pmf_cap)

  Get the 802.11w capability for the given interface.
- int qcsapi_SSID_set_pmf (const char *ifname, const qcsapi_SSID SSID_str, int pmf_cap)

  Set the 802.11w / PMF capability for the given interface.
- int qcsapi_SSID_get_wps_SSID (const char *ifname, qcsapi_SSID wps_SSID)

  Get the SSID associated with the WPS session.

9.15.1 Detailed Description

Parameters configured in a service set include encryption modes, authentication mode, pre-shared keys (PSK) and the passphrase. Thus these APIs mirror the "WPA" APIs, with the exception that the SSID APIs require a service set identifier (SSID). Two additional APIs verify an SSID is present in the configuration file and create a new service set.

Note

All SSID APIs work with the WPA Supplicant security configuration file, wpa_supplicant.conf. This file's location is determined by the get file path configuration API (section File Path configuration). Results from these APIs may be inconsistent or incorrect if the file path to the security configuration files has not been correctly configured.

9.15.2 Error Codes from SSID APIs

Two failure conditions are restricted to the SSID APIs.

The first failure condition is if the referenced SSID is not present in the configuration file. The error code in this situation will be:

```
-qcsapi_SSID_not_found
```

and the error message from the `call_qcsapi` scripting interface will be:

```
QCS API error 1002:  SSID not found
```

Use the Verify SSID API (qcsapi_SSID_verify_SSID) to confirm an SSID is present in the configuration file.

The second failure condition is if the referenced SSID is present in the configuration file, but a required parameter is not present.

An example is calling the SSID Get (Security) Protocol for an SSID that is configured without security.

Here the error code will be:

```
-qcsapi_SSID_parameter_not_found
```

and the error message from the `call_qcsapi` scripting interface will be:

```
QCS API error 1012:  Required parameter not found in the SSID configuration
block.
```

9.15.3 Function Documentation

9.15.3.1 qcsapi_SSID_create_SSID()

```
int qcsapi_SSID_create_SSID (
            const char * ifname,
            const qcsapi_SSID new_SSID )
```

Create a new SSID configuration, as identified by new_SSID.

If the Service Set is already present, this API returns an error.

The SSID must be a string with between 1 and 32 characters, as outlined in SSID_RULES

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
new_SSID	the new SSID to add.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi create_SSID <WiFi interface> <new SSID>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.2 qcsapi_SSID_remove_SSID()

```
int qcsapi_SSID_remove_SSID (
            const char * ifname,
            const qcsapi_SSID del_SSID )
```

Remove an existing SSID configuration, as identified by del_SSID.

If the Service Set is absent, this API returns an error.

Parameters

ifname	the interface to remove the SSID from.
del_SSID	the existing SSID to remove.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi remove_SSID <WiFi interface> <del SSID>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.3 qcsapi_SSID_verify_SSID()

```
int qcsapi_SSID_verify_SSID (
            const char * ifname,
            const qcsapi_SSID current_SSID )
```

Verifies a Service Set configuration is present, as identified by current_SSID. If the Service Set is not present, `-qcsapi_SSID_not_found` is returned (see Error Codes from SSID APIs for more details).

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to check.

Returns

0 if the SSID is present.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi verify_SSID <WiFi interface> <current SSID>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.4 qcsapi_SSID_rename_SSID()

```
int qcsapi_SSID_rename_SSID (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const qcsapi_SSID new_SSID )
```

Renames an SSID, as identified by current_SSID. If the original Service Set ID is not present, this API returns an error.

Both new_SSID and current_SSID must be strings with between 1 and 32 characters, as outlined in SSID_RULES

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	a currently defined SSID on this interface.
new_SSID	the new SSID value.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi rename_SSID <WiFi interface> <current SSID> <new SSID>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.5 qcsapi_SSID_get_SSID_list()

```
int qcsapi_SSID_get_SSID_list (
            const char * ifname,
            const unsigned int arrayc,
            char ** list_SSID )
```

Get the list of configured SSIDs.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
arrayc	the maximum number of SSID names to return
list_SSID	a pointer to the buffer for storing the returned values

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_SSID_list <WiFi interface>
```

The output will be the SSID list unless an error occurs. An additional optional parameter selects the number of SSID names to be displayed. The default count is 2; the maximum count is 10.

9.15.3.6 qcsapi_SSID_set_protocol()

```
int qcsapi_SSID_set_protocol (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const char * new_protocol )
```

Set the security authentication protocol (WPA or 11i or both) for an SSID. Valid values for new_protocol are WPA, 11i and WPAand11i.

This API is the SSID/STA equivalent of the AP only set beacon API.

Basic will not be accepted for the new protocol. To disable security for an SSID, use the SSID set authentication mode API (qcsapi_SSID_set_authentication_mode) with an authentication mode of NONE.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	an previously defined SSID to apply the protocol to.
new_protocol	the new protocol, as a string. See the **authentication protocol** table in Security definitions for valid values.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_set_proto <WiFi interface> <SSID name> <protocol type>
```

Unless an error occurs, the output will be the string `complete`.

Protocol type needs to be one as listed in the **authentication protocol** table in Security definitions.

See also

qcsapi_SSID_set_authentication_mode

qcsapi_wifi_set_beacon_type

9.15.3.7 qcsapi_SSID_get_protocol()

```
int qcsapi_SSID_get_protocol (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            string_16 current_protocol )
```

Get the security protocol (WPA or 11i or both) for an SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to check against.
current_protocol	set to one of the values in the **authentication protocol** table in /ref CommonSecurityDefinitions

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

This API is the SSID/STA equivalent of the get beacon API.

Note

This API should not be used to determine whether security is enabled for a particular SSID. Use the SSID Get Authentication Mode API to determine if security is enabled for the SSID. If the returned value is None, then security is disabled for the targeted SSID.

`call_qcsapi` interface:

```
call_qcsapi SSID_get_proto <WiFi interface> <SSID name>
```

Unless an error occurs, the response will be one of the values from the **authentication protocol** table in Security definitions

See also

qcsapi_SSID_get_authentication_mode

qcsapi_wifi_get_beacon_type

9.15.3.8 qcsapi_SSID_get_encryption_modes()

```
int qcsapi_SSID_get_encryption_modes (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            string_32 encryption_modes )
```

Get available encryption modes for an SSID.

This API is called to determing the encryption modes supported on the given SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to read the encryption modes from.
encryption_modes	a comma delimited set of strings, one for each encryption mode. The values in this string are from the **encryption** type table in Security definitions.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_get_encryption_modes <WiFi interface> <SSID name>
```

Unless an error occurs, the output will be one of these three strings, AESEncryption, TKIPEncryption or TKIPand↩ AESEncryption. That is, one of the values from the **encryption** type table in Security definitions.

See also

qcsapi_SSID_set_encryption_modes

9.15.3.9 qcsapi_SSID_set_encryption_modes()

```
int qcsapi_SSID_set_encryption_modes (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const string_32 encryption_modes )
```

Configure available encryption modes for an SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to set the encryption mode against.
encryption_modes	a value as per the **encryption** type table in Security definitions.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

`call_qcsapi SSID_set_encryption_modes <WiFi interface> <SSID name> <encryption mode(s)>` where <encryption mode(s)> is one of the modes listed in the **encryption** type table in Security definitions.

Unless an error occurs, the output will be the string `complete`.

9.15.3.10 qcsapi_SSID_get_group_encryption()

```
int qcsapi_SSID_get_group_encryption (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            string_32 encryption_mode )
```

Get the group encryption cipher for an SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID for which to retrieve the group encryption cipher
encryption_mode	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

This command is only supported on Stations.

`call_qcsapi SSID_get_group_encryption <WiFi interface> <SSID>`

The output will be the encryption cipher unless an error occurs.

9.15.3.11 qcsapi_SSID_set_group_encryption()

```
int qcsapi_SSID_set_group_encryption (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const string_32 encryption_mode )
```

Set the group encryption cipher for an SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID for which to set the group encryption cipher
encryption_mode	the cipher to be applied

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

This command is only supported on Stations.

`call_qcsapi` interface:

```
call_qcsapi SSID_set_group_encryption <WiFi interface> <SSID> <encryption↩
_mode>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.12 qcsapi_SSID_get_authentication_mode()

```
int qcsapi_SSID_get_authentication_mode (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            string_32 authentication_mode )
```

Get the current configured authentication mode for an SSID.

This API is the SSID/STA version of WPA get authentication mode API (qcsapi_wifi_get_WPA_authentication_↩ mode); see that section for a description of possible authentication modes returned by this API. If security is disabled for the referenced SSID, this API will return `NONE`.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to get the authentication modes from.
authentication_mode	return parameter to store the authentication type.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_get_authentication_mode <WiFi interface> <SSID name>
```

Unless an error occurs, the response will be of `PSKAuthentication, EAPAuthentication` or `NONE`. That is, one of the values outlined in the **authentication type** table in Security definitions.

A response of NONE implies security is disabled for the referenced SSID.

See also

qcsapi_wifi_get_WPA_authentication_mode

9.15.3.13 qcsapi_SSID_set_authentication_mode()

```
int qcsapi_SSID_set_authentication_mode (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const string_32 authentication_mode )
```

Set the authentication mode for an SSID.

This API is the SSID/STA version of WPA set authentication mode API (qcsapi_wifi_set_WPA_authentication_↩ mode);

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to get the authentication modes from.
authentication_mode	the authentication mode to use. One of the values from the **authentication type** table in Security definitions.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_set_authentication_mode <WiFi interface> <SSID name> <authentication mode>
```

Unless an error occurs, the output will be the string `complete`.

Valid values for the authentication mode paramter are outlined in the **authentication type** table in Security definitions.

To disable authentication on the SSID, pass the value `NONE` to the authentication mode parameter.

See also

qcsapi_wifi_set_WPA_authentication_mode

9.15.3.14 qcsapi_SSID_get_pre_shared_key()

```
int qcsapi_SSID_get_pre_shared_key (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const qcsapi_unsigned_int key_index,
            string_64 pre_shared_key )
```

Get the WPA or RSN preshared key for an SSID.

Note

This API can only be used on the primary interface (wifi0)

This API can only be used in STA mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
current_SSID	the SSID for which to retrieve the preshared key
key_index	reserved - set to zero
pre_shared_key	a pointer to the buffer for storing the returned value

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_get_pre_shared_key <WiFi interface> <SSID> <key index>
```

The output will be the preshared key unless an error occurs.

9.15.3.15 qcsapi_SSID_set_pre_shared_key()

```
int qcsapi_SSID_set_pre_shared_key (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const qcsapi_unsigned_int key_index,
            const string_64 pre_shared_key )
```

Set the WPA or RSN preshared key for an SSID.

Note

This API can only be used on the primary interface (wifi0)

This API can only be used in STA mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
current_SSID	the SSID to set the WPA PSK on
key_index	reserved - set to zero
pre_shared_key	a 64 hex digit PSK

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_set_pre_shared_key <WiFi interface> <SSID> <key index>
<preshared key>
```

Unless an error occurs, the output will be the string `complete`.

9.15.3.16 qcsapi_SSID_get_key_passphrase()

```
int qcsapi_SSID_get_key_passphrase (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const qcsapi_unsigned_int key_index,
            string_64 passphrase )
```

Get the passphrase for an SSID.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to get the passphrase for.
key_index	reserved - set to zero
passphrase	return parameter to contain the NULL terminated passphrase.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_get_key_passphrase <WiFi interface> <SSID name> 0

call_qcsapi SSID_get_passphrase <WiFi interface> <SSID name> 0
```

Unless an error occurs, the output will be the passphrase configured for the SSID.

9.15.3.17 qcsapi_SSID_set_key_passphrase()

```
int qcsapi_SSID_set_key_passphrase (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            const qcsapi_unsigned_int key_index,
            const string_64 passphrase )
```

Set the ASCII passphrase for a Service Set on a STA

By the WPA standard, the passphrase is required to have between 8 and 63 ASCII characters.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID to set the passphrase on.
key_index	reserved - set to zero
the	NULL terminated passphrase string, 8 - 63 ASCII characters (NULL termination not included in the count)

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi SSID_set_key_passphrase <WiFi interface> <SSID name> 0 <new
passphrase>

call_qcsapi SSID_set_passphrase <WiFi interface> <SSID name> 0 <new passphrase>
```

Unless an error occurs, the output will be the string `complete`.

Note

The Linux shell processes the passphase parameter. Selected characters are interpreted by the shell, including the dollar sign ($), the backslash () and the backquote (`). We recommend putting the new passphrase in quotes and/or using the backslash character to escape characters that could be processed by the shell.

9.15.3.18 qcsapi_wifi_update_bss_cfg()

```
int qcsapi_wifi_update_bss_cfg (
            const char * ifname,
            const qcsapi_wifi_mode wifi_mode,
            const char * ssid,
            const char * param_name,
            const char * param_value,
            const char * param_type )
```

Note

The parameter will not immediately take effect until the deamon reconfigured.

The API directly manipulate the secruity deamon configuration file.

Please make sure the parameters are valid.

Parameters

ifname	the interface to perform the action on, for AP mode only
mode	AP or STA
ssid	an previously defined SSID to apply the param to, for STA mode only
param_name	specific the paramter name
param_value	specific the paramter value For parameters other than "ssid", the value of "NULL" stands to delete the parameter Parameter of "ssid" will be deleted if the value is not identical to param ssid
param_type	specific the paramter attribute

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi update_bss_cfg <WiFi interface> <mode> <bss_id> <param_name>
<parram_value> [param_type]
```

Unless an error occurs, the output will be the current PMF capability.

Example of adding or updating an access point paramter:
`call_qcsapi update_bss_cfg wifi0 ap <ifname> <param_name> <param_value>`
Example of deleting an access point paramter:
`call_qcsapi update_bss_cfg wifi0 ap <ifname> <param_name> <"NULL">`

Example of adding or updating a station paramter:
`call_qcsapi update_bss_cfg wifi0 sta <ssid> <param_name> <param_value> [ 0 | 1 ]`
Example of deleting a station paramter:
`call_qcsapi update_bss_cfg wifi0 sta <ssid> <param_name> <"NULL"> [ 0 | 1 ]`

9.15.3.19 qcsapi_SSID_get_pmf()

```
int qcsapi_SSID_get_pmf (
            const char * ifname,
            const qcsapi_SSID current_SSID,
            int * p_pmf_cap )
```

Returns the current 802.11w / PMF capability.

Note

staonly

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID on which to get the PMF capability

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

`call_qcsapi SSID_get_pmf <WiFi interface> <SSID>`

Unless an error occurs, the output will be the current PMF capability.

See also

qcsapi_SSID_get_pmf

9.15.3.20 qcsapi_SSID_set_pmf()

```
int qcsapi_SSID_set_pmf (
            const char * ifname,
            const qcsapi_SSID SSID_str,
            int pmf_cap )
```

Sets the 802.11w / PMF capability.

Note

staonly

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_SSID	the SSID on which to set the PMF capability
pmf_cap.	

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

`call_qcsapi SSID_set_pmf <WiFi interface> <SSID> { 0 | 1 | 2 }`

Unless an error occurs, the output will be the string `complete`.

The final '0' in the command line represents the key index.

Note

The Linux shell processes the PMF parameter

See also

qcsapi_SSID_set_pmf

9.15.3.21 qcsapi_SSID_get_wps_SSID()

```
int qcsapi_SSID_get_wps_SSID (
            const char * ifname,
            qcsapi_SSID wps_SSID )
```

This API returns the SSID as configured via WPS. This network block is marked using a 'flags' parameter to indicate that the SSID was configured via WPS.

Note

This API can only be used in STA mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wps_SSID	the return value to contain the SSID.

Returns

>= 0 on success, -qcsapi_configuration_error if the SSID is not one configured via WPS, -qcsapi_only_on↩ _STA if called on an AP, -EFAULT or -qcsapi_programming_error on other errors (NULL parameter etc).

`call_qcsapi` interface:

```
call_qcsapi SSID_get_WPS_SSID <WiFi interface>
```

Unless an error occurs, the output will be the string containing the SSID as obtained via WPS.

9.16 WPS APIs

Functions

- int qcsapi_wps_registrar_report_button_press (const char *ifname)

  WPS registrar report button press.
- int qcsapi_wps_registrar_report_pin (const char *ifname, const char *wps_pin)

  Report a PIN event on the registrar.
- int qcsapi_wps_registrar_get_pp_devname (const char *ifname, int blacklist, string_128 pp_devname)

  Get the WPS Access Control list.
- int qcsapi_wps_registrar_set_pp_devname (const char *ifname, int update_blacklist, const string_256 pp_↩ devname)

  Set WPS Access Control Device ID List.
- int qcsapi_wps_enrollee_report_button_press (const char *ifname, const qcsapi_mac_addr bssid)

  Report a WPS PBC event on the enrollee.
- int qcsapi_wps_enrollee_report_pin (const char *ifname, const qcsapi_mac_addr bssid, const char *wps_↩ pin)

  Report a PIN event on the enrollee.
- int qcsapi_wps_enrollee_generate_pin (const char *ifname, const qcsapi_mac_addr bssid, char *wps_pin)

  Generate a WPS PIN on the enrollee.
- int qcsapi_wps_get_ap_pin (const char *ifname, char *wps_pin, int force_regenerate)

  Get the AP PIN used for WPS PIN operation.
- int qcsapi_wps_set_ap_pin (const char *ifname, const char *wps_pin)

  set the AP PIN used for WPS PIN operation .
- int qcsapi_wps_save_ap_pin (const char *ifname)

  save ap PIN to configure file
- int qcsapi_wps_enable_ap_pin (const char *ifname, int enable)

  enable/disable ap pin function
- int qcsapi_wps_get_sta_pin (const char *ifname, char *wps_pin)

  Generate a new PIN randomly.
- int qcsapi_wps_get_state (const char *ifname, char *wps_state, const qcsapi_unsigned_int max_len)

  Get the state of the current WPS session.
- int qcsapi_wps_get_configured_state (const char *ifname, char *wps_state, const qcsapi_unsigned_int max_len)

  Get the WPS configured state for the given interface.
- int qcsapi_wps_get_runtime_state (const char *ifname, char *state, int max_len)

  Get the WPS runtime state for the given interface.
- int qcsapi_wps_set_configured_state (const char *ifname, const qcsapi_unsigned_int state)

  Set the WPS configured state for the given interface.
- int qcsapi_wps_get_param (const char *ifname, qcsapi_wps_param_type wps_type, char *wps_str, const qcsapi_unsigned_int max_len)

  Get a WPS parameter.
- int qcsapi_wps_set_timeout (const char *ifname, const int value)

  set wps walk time value from 120s to 600s
- int qcsapi_wps_on_hidden_ssid (const char *ifname, const int value)

  set wps_on_hidden_ssid enabled or disabled
- int qcsapi_wps_on_hidden_ssid_status (const char *ifname, char *state, int max_len)

  get wps_on_hidden_ssid status
- int qcsapi_wps_upnp_enable (const char *ifname, const int value)

  enable or disable wps upnp module
- int qcsapi_wps_upnp_status (const char *ifname, char *reply, int reply_len)

get upnp status

- int qcsapi_wps_allow_pbc_overlap (const char *ifname, const qcsapi_unsigned_int allow)

  Allow or forbid the detection of WPS PBC overlap.
- int qcsapi_wps_get_allow_pbc_overlap_status (const char *ifname, int *status)

  get status if PBC overlap is allowed on AP or STA.
- int qcsapi_wps_set_access_control (const char *ifname, uint32_t ctrl_state)

  Enable/Disable the WPS Pair Protection for the given interface.
- int qcsapi_wps_get_access_control (const char *ifname, uint32_t *ctrl_state)

  Get the WPS Pair Protection state for the given interface.
- int qcsapi_wps_set_param (const char *ifname, const qcsapi_wps_param_type param_type, const char *param_value)

  Set a WPS parameter.
- int qcsapi_wps_cancel (const char *ifname)

  Cancel the ongoing wps procedure if any.
- int qcsapi_wps_set_pbc_in_srcm (const char *ifname, const qcsapi_unsigned_int enabled)

  Add/remove PBC methods in SRCM.
- int qcsapi_wps_get_pbc_in_srcm (const char *ifname, qcsapi_unsigned_int *p_enabled)

  Get currently setting of PBC methods in SRCM attribute.
- int qcsapi_registrar_set_default_pbc_bss (const char *ifname)

  set default bss for WPS Push button
- int qcsapi_registrar_get_default_pbc_bss (char *default_bss, int len)

  get default bss for WPS Push button
- int qcsapi_wps_set_default_pbc_bss (const char *ifname)

  set default interface for WPS Push button
- int qcsapi_wps_get_default_pbc_bss (char *default_bss, int len)

  get the associated interface on first radio device for WPS Push button

9.16.1 Detailed Description

9.16.2 Overview

Under the WPS standard, a WiFi device can be either a Registrar or an Enrollee. In this context, currently an AP is always a Registrar and a STA is always an Enrollee.

9.16.3 Function Documentation

9.16.3.1 qcsapi_wps_registrar_report_button_press()

```
int qcsapi_wps_registrar_report_button_press (
            const char * ifname )
```

This API starts a WPS session on the Registrar (AP) by pressing the (virtual) WPS Push Button.

Under the WPS standard, a WPS session can be started by pressing a virtual button; i.e. by entering a command.

A side effect of this API call is that a WPS session will be started.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_report_button_press <WiFi interface>

call_qcsapi registrar_report_pbc <WiFi interface>
```

This API has 2 scripting interface synonyms.

Unless an error occurs, the output will be the string `complete`.

9.16.3.2 qcsapi_wps_registrar_report_pin()

```
int qcsapi_wps_registrar_report_pin (
            const char * ifname,
            const char * wps_pin )
```

This API starts a WPS session on the registrar (AP) by reporting a PIN event.

Under the WPS standard, a WPS session can be started by entering a PIN.

The PIN is a sequence of either 4 or 8 digits. If the proposed PIN has a length different from 4 or 8 characters, or if any of the characters are not digits, the API will return an error code of Invalid Value (-EINVAL).

Note

The 8 digit PIN (which has a checksum) does not check the validity of the checksum digit.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wps_pin	the NULL terminated PIN - either 4 or 8 decimal numbers.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_report_pin <WiFi interface> <PIN>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.3 qcsapi_wps_registrar_get_pp_devname()

```
int qcsapi_wps_registrar_get_pp_devname (
            const char * ifname,
            int blacklist,
            string_128 pp_devname )
```

Get the WPS Access Control list.

Note

This API is only relevant on an AP device

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
pp_devname	comma-separated list of device IDs allowed or denied to receive credentials via WPS

Returns

0 if the command succeeded and pp_devname contains list of Device IDs allowed

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_get_pp_devname <WiFi interface> [blacklist]
```

Unless an error occurs, the output will be the string of allowed Device IDs.

9.16.3.4 qcsapi_wps_registrar_set_pp_devname()

```
int qcsapi_wps_registrar_set_pp_devname (
            const char * ifname,
            int update_blacklist,
            const string_256 pp_devname )
```

Set the list of Device IDs that are allowed or denied to receive WPS credentials from the AP.

Note

This API can only be used in AP mode.

The Device IDs are a comma separated list 1 to 256 characters in length with commas as delimiters

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
update_blacklist	flag to indicate whether update white-list or black-list
pp_devname	comma-separated list of device IDs allowed or denied to receive credentials via WPS.

Returns

0 if the command succeeded and the SSID is updated.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_set_pp_devname <WiFi interface> [blacklist] <Comma-separated
list of device ID>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.5 qcsapi_wps_enrollee_report_button_press()

```
int qcsapi_wps_enrollee_report_button_press (
            const char * ifname,
            const qcsapi_mac_addr bssid )
```

This API starts a WPS session on the Enrollee (STA) by pressing the (virtual) WPS Push Button.

Under the WPS standard, a WPS session can be started by pressing a virtual button; i.e. by entering a command.

The bssid parameter is present for future expansion and should be set to all 0s (zeros).

`call_qcsapi` interface:

```
call_qcsapi enrollee_report_button_press <WiFi interface>call_qcsapi enrollee↩
_report_pbc <WiFi interface>
```

This API has 2 scripting interface synonyms. The bssid parameter is not required and will default to all zeros.

Unless an error occurs, the output will be the string `complete`.

9.16.3.6 qcsapi_wps_enrollee_report_pin()

```
int qcsapi_wps_enrollee_report_pin (
            const char * ifname,
            const qcsapi_mac_addr bssid,
            const char * wps_pin )
```

This API starts a WPS session on the enrollee (STA) by reporting a PIN event.

Under the WPS standard, a WPS session can be started by entering a PIN.

The PIN is a sequence of either 4 or 8 digits. If the proposed PIN has a length different from 4 or 8 characters, or if any of the characters are not digits, the API will return an error code of Invalid Value (-EINVAL).

Note

The 8 digit PIN (which has a checksum) does not check the validity of the checksum digit.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
bssid	the BSSID to report the PIN evens for.
wps_pin	the NULL terminated PIN - either 4 or 8 decimal numbers.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enrollee_report_pin <WiFi interface> <PIN>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.7 qcsapi_wps_enrollee_generate_pin()

```
int qcsapi_wps_enrollee_generate_pin (
            const char * ifname,
            const qcsapi_mac_addr bssid,
            char * wps_pin )
```

This API starts a WPS session on the enrollee (STA) by generating a PIN and then reporting that newly generated PIN to any suitably configured and available registrars. The generated PIN will have 8 digits.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
bssid	reserved - set to all zeros.
wps_pin	return parameter to contain the WPS PIN (8 digits, so the string should be at least 9 bytes long).

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enrollee_generate_pin <WiFi interface>
```

Unless an error occurs, the output will be a string of 8 digits - the newly generated PIN

The bssid parameter is not required and will default to all zeros.

9.16.3.8 qcsapi_wps_get_ap_pin()

```
int qcsapi_wps_get_ap_pin (
            const char * ifname,
            char * wps_pin,
            int force_regenerate )
```

This API call is used to get the AP PIN associated with the WPS PIN function. The PIN is either 4 digits or 8 digits long.

Note

this API can only be called on an AP device.

Parameters

ifname	each interface in MBSS
wps_pin	return parameter for containing the NULL terminated string.
force_regenerate	whether to force the AP daemon (hostapd) to regenerate the WPS PIN - a random PIN - for this call.

Note

a side effect of this call is if there is no WPS PIN currently set for the device, a random PIN will be generated.

Returns

>= 0 success, < 0 or -qcsapi_only_on_AP on error. If success, the wps_pin parameter will be filled with the NULL terminated string containing the PIN.

`call_qcsapi` interface:

```
call_qcsapi get_wps_ap_pin <WiFi interface>
```

Unless an error occurs, the output will be a string of 8 digits - the PIN on the AP.

9.16.3.9 qcsapi_wps_set_ap_pin()

```
int qcsapi_wps_set_ap_pin (
            const char * ifname,
            const char * wps_pin )
```

This API call is used to set the AP PIN associated with the WPS PIN function. The PIN is either 4 digits or 8 digits long.

Note

this API can only be called on an AP device.

Parameters

ifname	each interface in MBSS
wps_pin	return parameter for containing the NULL terminated string.

Returns

>= 0 success, < 0 or -qcsapi_only_on_AP on error. **`call_qcsapi` interface:**

```
call_qcsapi set_wps_ap_pin <WiFi interface> <wps_pin>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.10 qcsapi_wps_save_ap_pin()

```
int qcsapi_wps_save_ap_pin (
            const char * ifname )
```

This API call is used to save PIN to configure file

Note

this API can only be called on an AP device.

Parameters

ifname	each interface in MBSS

Returns

>= 0 success, < 0 or -qcsapi_parameter_not_found on error. **`call_qcsapi` interface:**

```
call_qcsapi save_wps_ap_pin <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.11 qcsapi_wps_enable_ap_pin()

```
int qcsapi_wps_enable_ap_pin (
            const char * ifname,
            int enable )
```

This API call is used to enable/disable external registrar configure this AP when wps state is not configured

Note

this API can only be called on an AP device.

Parameters

ifname	each interface in MBSS
enable	

Returns

>= 0 success, < 0 or -qcsapi_parameter_not_found on error. **`call_qcsapi` interface:**

```
call_qcsapi enable_wps_ap_pin <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.12 qcsapi_wps_get_sta_pin()

```
int qcsapi_wps_get_sta_pin (
            const char * ifname,
            char * wps_pin )
```

This API is used to generate a new PIN randomly on a STA. This API won't start WPS session.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wps_pin	return parameter for containing the NULL terminated string.

Returns

>= 0 success, < 0 on error. If success, the wps_pin parameter will be filled with the NULL terminated string containing the PIN.

`call_qcsapi` interface:

```
call_qcsapi get_wps_sta_pin <WiFi interface>
```

Unless an error occurs, the output will be a string of 8 digits.

9.16.3.13 qcsapi_wps_get_state()

```
int qcsapi_wps_get_state (
            const char * ifname,
            char * wps_state,
            const qcsapi_unsigned_int max_len )
```

Get the current WPS state, reported as a string in the format "%d (%s)", that is, an integer followed by a short descriptive string in parentheses. This API works for either an enrollee or a registrar; or stated differently, it works on both an AP and a STA.

Possible WPS states are:

- 0 (WPS_INITIAL) - Initial WPS state.
- 1 (WPS_START) - WPS transaction has started.
- 2 (WPS_SUCCESS) - WPS transaction succeeded and the device is in association with its partner.
- 3 (WPS_ERROR) - WPS transaction ended with an error.
- 4 (WPS_TIMEOUT) - WPS transaction timed out.
- 5 (WPS_OVERLAP) - WPS overlap is detected.
- 6 (WPS_M2_SEND) - WPS is sending M2 frame.
- 7 (WPS_M8_SEND) - WPS is sending M8 frame.
- 8 (WPS_STA_CANCEL) - WPS is canceled by STA.
- 9 (WPS_STA_PIN_ERR) - WPS fail for wrong pin from STA.
- 10 (WPS_AP_PIN_SUC) - WPS AP pin success.
- 11 (WPS_AP_PIN_ERR) - WPS AP pin fail.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wps_state	return parameter for storing the informative WPS state string.
max_len	the length of the wps_state string passed in.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_wps_state <WiFi interface>
```

Unless an error occurs, the output will be one of the WPS State strings listed in the description of the API itself.

9.16.3.14 qcsapi_wps_get_configured_state()

```
int qcsapi_wps_get_configured_state (
            const char * ifname,
            char * wps_state,
            const qcsapi_unsigned_int max_len )
```

This API call is used to find the WPS configured state - either configured or not configured.

Note

this API can only be called on an AP device. WPS configured/not configured is a concept that only applies to the AP.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wps_state	return parameter to store the WPS state (configured or not configured).
max_len	the size of the input buffer (wps_state).

Returns

>= 0 on success, < 0 on error. If success, the wps_state parameter will be filled with the NULL terminated string 'configured' or 'not configured'.

`call_qcsapi` interface:

```
call_qcsapi get_wps_configured_state <WiFi interface>
```

Unless an error occurs, the output will be the string '`configured`' or '`not configured`'.

9.16.3.15 qcsapi_wps_get_runtime_state()

```
int qcsapi_wps_get_runtime_state (
            const char * ifname,
            char * state,
            int max_len )
```

This API call is used to find the WPS runtime state, disabled, not configured or configuired

Note

this API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
state	return parameter to store the WPS state (disabled, configured or not configured).
max_len	the size of the input buffer (state).

Returns

>= 0 on success, < 0 on error. If success, the wps_state parameter will be filled with the NULL terminated string 'disable', 'configured' or 'not configured'.

`call_qcsapi` interface:

```
call_qcsapi get_wps_runtime_state <WiFi interface>
```

Unless an error occurs, the output will be the string '`disabled`', '`configured`' or '`not configured`'.

9.16.3.16 qcsapi_wps_set_configured_state()

```
int qcsapi_wps_set_configured_state (
            const char * ifname,
            const qcsapi_unsigned_int state )
```

This API call is used to set the WPS state to configured or unconfigured.

Note

This API can only be called on an AP.

If Hotspot 2.0 is enabled then WPS configuration will not take any effect.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
state	either 0 (disabled), 1 (not configured) or 2 (configured).

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi set_wps_configured_state <WiFi interface> <0 | 1 | 2>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.17 qcsapi_wps_get_param()

```
int qcsapi_wps_get_param (
            const char * ifname,
            qcsapi_wps_param_type wps_type,
            char * wps_str,
            const qcsapi_unsigned_int max_len )
```

This API returns the value of a WPS Parameter.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0). \wifi1 etc.
wps_type	the WPS parameter. See the definition of the enun `qcsapi_wps_param_type`.
wps_str	Address of the string to receive the parameter's value.
max_len	Maximum number of characters that can be written to the parameter `wps_str`

Returns

>= 0 on success, < 0 on failure.

call_qcsapi interface:

```
call_qcsapi get_wps_param <WiFi interface> <WPS parameter name>
```

where WPS parameter name is one of uuid, os_version, device_name, config_methods or ap_setup_locked last_config_error registrar_number registrar_established force_broadcast_uuid third_party_band

Note

this API can be used both on AP and STA, except for parameter name is ap_setup_locked, third_party_band and force_broadcast_uuid.

Unless an error occurs, the output will be the value of the selected WPS parameter.

Note

last_config_error, registrar_number, and registrar_established are not supported currently.

9.16.3.18 qcsapi_wps_set_timeout()

```
int qcsapi_wps_set_timeout (
            const char * ifname,
            const int value )
```

This API call is used to set the wps walk time

Note

this API can be called both on an AP device or a STA device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value.	walk time value

Returns

>= 0 on success, < 0 on failure.

call_qcsapi interface:

```
call_qcsapi wps_set_timeout <WiFi interface> <timeout value>
```

Unless an error occurs, the output will be the string complete.

9.16.3.19 qcsapi_wps_on_hidden_ssid()

```
int qcsapi_wps_on_hidden_ssid (
            const char * ifname,
            const int value )
```

This API call is used to enable or disable the feature wps_on_hidden_ssid

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...
value.	1 wps_on_hidden_ssid enabled, 0 wps_on_hidden_ssid disabled

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi wps_on_hidden_ssid <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.20 qcsapi_wps_on_hidden_ssid_status()

```
int qcsapi_wps_on_hidden_ssid_status (
            const char * ifname,
            char * state,
            int max_len )
```

This API call is used to check status of wps_on_hidden_ssid

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...

Returns

"on", "off" or "FAIL"

`call_qcsapi` interface:

```
call_qcsapi wps_on_hidden_ssid_status <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.21 qcsapi_wps_upnp_enable()

```
int qcsapi_wps_upnp_enable (
            const char * ifname,
            const int value )
```

This API call is used to enable or disable wps upnp module

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value.	1 upnp enabled, 0 upnp disabled

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi wps_upnp_enable <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.22 qcsapi_wps_upnp_status()

```
int qcsapi_wps_upnp_status (
            const char * ifname,
            char * reply,
            int reply_len )
```

This API call is used to get upnp status

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
reply.	reply buffer
reply_len.	reply buffer length

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi wps_upnp_status <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.23 qcsapi_wps_allow_pbc_overlap()

```
int qcsapi_wps_allow_pbc_overlap (
            const char * ifname,
            const qcsapi_unsigned_int allow )
```

This API call is used to allow/forbid the detection of PBC overlap.

Note

this API can be called both on an AP device or a STA device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
allow.	1 indicates allow, 0 indicates forbid.

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi allow_pbc_overlap <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.24 qcsapi_wps_get_allow_pbc_overlap_status()

```
int qcsapi_wps_get_allow_pbc_overlap_status (
            const char * ifname,
            int * status )
```

This API returns the status if PBC overlap is allowed on AP or STA.

Note

this API can be called both on an AP device or a STA device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
status	the return value if allow PBC overlap.

Returns

>= 0 on success, < 0 on failure. If success, the status of allowing PBC overlap (0/1) will be returned.

`call_qcsapi` interface:

```
call_qcsapi get_allow_pbc_overlap_status <WiFi interface>
```

Unless an error occurs, the output will be the string '1' or '0'

9.16.3.25 qcsapi_wps_set_access_control()

```
int qcsapi_wps_set_access_control (
            const char * ifname,
            uint32_t ctrl_state )
```

This API call is used to Enable/Disable the WPS Pair Protection.

Note

this API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ctrl_state	either 0 (disabled) or 1 (enabled).

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi set_wps_access_control <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.26 qcsapi_wps_get_access_control()

```
int qcsapi_wps_get_access_control (
            const char * ifname,
            uint32_t * ctrl_state )
```

This API call is used to get the WPS Pair Protection state - either enabled or disabled.

Note

this API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ctrl_state	return parameter to store the WPS Pair Protection state (enabled or disabled).

Returns

>= 0 on success, < 0 on error. If success, the wps pair protection state parameter (0/1)will be returned.

`call_qcsapi` interface:

```
call_qcsapi get_wps_access_control <WiFi interface>
```

Unless an error occurs, the output will be the string '1' or '0'.

9.16.3.27 qcsapi_wps_set_param()

```
int qcsapi_wps_set_param (
            const char * ifname,
            const qcsapi_wps_param_type param_type,
            const char * param_value )
```

This API is called to set a WPS Parameter.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0). \wifi1 etc. use \all to set parameter for each existing interface, AP mode only.
wps_type	the WPS parameter. See the definition of the enun `qcsapi_wps_param_type`.
param_value	Address of the string to set the parameter's value.

Returns

>= 0 on success, < 0 on failure.

`call_qcsapi` interface:

```
call_qcsapi set_wps_param <WiFi interface> <WPS parameter name> <WPS parameter
value>
```

where `WPS parameter name` is one of `config_methods`, `ap_pin`, `setup_lock`, `ap_setup_locked`, `third_party_band`, `uuid` or `force_broadcast_uuid`. The API is only available for AP mode, except for the parameter name `uuid` and `config_methods`.

Parameter `ap_setup_locked` can only be set or reset when the WPS parameter `ap_pin_fail_method` is set to auto_lockdown.

When parameter name is `config_methods`, the available parameter value is one of following value or combination of them, `usba`, `ethernet`, `label`, `display`, `ext_nfc_token`, `int_nfc_token`, `nfc_interface`, `push_button`, `keypad`, `virtual_display`, `virtual_push_button`, `physical_push_button`.

The parameter value of uuid has format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. For example:
`cdcb13e6-baa5-5f43-807f-4b4c28223a68`

Unless an error occurs, the output will be the string `complete`.

9.16.3.28 qcsapi_wps_cancel()

```
int qcsapi_wps_cancel (
            const char * ifname )
```

This API equivalent to "wpa_cli wps_cancel". It will cancel ongoing wps procedure, and do nothing if there are no wps procedure undergoing.

Note

this API can only be called on an STA device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wps_cancel <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.29 qcsapi_wps_set_pbc_in_srcm()

```
int qcsapi_wps_set_pbc_in_srcm (
            const char * ifname,
            const qcsapi_unsigned_int enabled )
```

This API is used to add or remove PBC methods in SRCM (selected registrar config methods) attribute in WSC IE.

Note

this API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enabled	1 to add and 0 to remove PBC methods in SRCM attribute in WSC IE.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_wps_pbc_in_srcm <WiFi interface> <1 | 0>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.30 qcsapi_wps_get_pbc_in_srcm()

```
int qcsapi_wps_get_pbc_in_srcm (
            const char * ifname,
            qcsapi_unsigned_int * p_enabled )
```

This API is used to get currently setting of PBC methods in SRCM attribute.

Note

this API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_enabled	Where to store the result return.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_wps_pbc_in_srcm <WiFi interface>
```

Unless an error occurs, the output will be the string `0` or `1`.

9.16.3.31 qcsapi_registrar_set_default_pbc_bss()

```
int qcsapi_registrar_set_default_pbc_bss (
            const char * ifname )
```

This API is used to set default bss for WPS Push button if there's more than one BSS such like MBSS mode default bss for WPS PBC is primary interface(wifi0) after powered up

Note

this API can only be called on an AP device. set "null" would remove default setting

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_set_default_pbc_bss <WiFi interface | null>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.32 qcsapi_registrar_get_default_pbc_bss()

```
int qcsapi_registrar_get_default_pbc_bss (
            char * default_bss,
            int len )
```

This API is used to get default bss for WPS Push button default bss for WPS PBC is primary interface(wifi0) after powered up

Note

this API can only be called on an AP device.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi registrar_get_default_pbc_bss
```

Unless an error occurs, the output will be WiFi interface or null.

9.16.3.33 qcsapi_wps_set_default_pbc_bss()

```
int qcsapi_wps_set_default_pbc_bss (
            const char * ifname )
```

This API is used to associate a wireless interface with WPS Push button if there's more than one wireless interface in the system, such as repeater mode. default interface for WPS PBC is primary interface(wifi0) after powered up

Note

This API can be called under AP, STA or Repeater modes

Parameters

ifname	the interface to perform the action on. wifiX, For X=0,1,...

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wps_set_default_pbc_bss <WiFi interface | null>
```

Unless an error occurs, the output will be the string `complete`.

9.16.3.34 qcsapi_wps_get_default_pbc_bss()

```
int qcsapi_wps_get_default_pbc_bss (
            char * default_bss,
            int len )
```

This API is used to get default interface on first radio for WPS Push button default interface for WPS PBC is primary interface on first radio device after powered up

Note

This API can be called under AP, STA or Repeater modes

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi wps_get_default_pbc_bss
```

Unless an error occurs, the output will be an interface name

9.17 LED and GPIO APIs

Although the APIs make a formal distinction between LEDs and GPIO pins, currently all LEDs are controlled thru GPIO pins.
To accommodate different board designs, all LEDs / GPIO pins must be configured prior to use by software. The configuration is persistent and should only need to be done once. GPIO pin configuration is one of:

Typedefs

- typedef void(∗ reset_device_callback) (uint8_t reset_device_pin, uint8_t current_level)

Functions

- int qcsapi_gpio_set_config (const uint8_t gpio_pin, const qcsapi_gpio_config new_gpio_config)

  Set GPIO configuration.
- int qcsapi_gpio_get_config (const uint8_t gpio_pin, qcsapi_gpio_config ∗p_gpio_config)

  Get GPIO configuration.
- int qcsapi_led_get (const uint8_t led_ident, uint8_t ∗p_led_setting)

  Get LED state.
- int qcsapi_led_set (const uint8_t led_ident, const uint8_t new_led_setting)

  Set LED state.
- int qcsapi_led_pwm_enable (const uint8_t led_ident, const uint8_t onoff, const qcsapi_unsigned_int high_↩ count, const qcsapi_unsigned_int low_count)

  Enable pulse wide modulation for LED GPIO pin.
- int qcsapi_led_brightness (const uint8_t led_ident, const qcsapi_unsigned_int level)

  Set LED brightness level.
- int qcsapi_gpio_monitor_reset_device (const uint8_t reset_device_pin, const uint8_t active_logic, const int blocking_flag, reset_device_callback respond_reset_device)

  Monitor the reset device GPIO pin.
- int qcsapi_gpio_enable_wps_push_button (const uint8_t wps_push_button, const uint8_t active_logic, const uint8_t use_interrupt_flag)

  Enable the WPS GPIO push button.

9.17.1 Detailed Description

- qcsapi_gpio_not_available = 0
- qcsapi_gpio_input_only = 1
- qcsapi_gpio_output = 2

Default configuration is qcsapi_gpio_not_available. A pin configured for output can be read as input.
All GPIO pins are accessed through the LED APIs, including GPIO pins that do not control an LED. An LED / GPIO pin can be either HIGH (value for setting is 1) or low (value for setting is 0). Be aware that a particular LED / GPIO pin can either be active high or active low. Consult the board documentation or schematics for details on the logic for each GPIO pin.
GPIO pin numbers range from 0 to 31.

9.17.2 Typedef Documentation

9.17.2.1 reset_device_callback

```
typedef void(* reset_device_callback) (uint8_t reset_device_pin, uint8_t current_level)
```

This typedef is used to force a function prototype for reset button function callback.

The reset_device_pin passes in the GPIO pin being monitored, and the current_level is either 1 or 0.

See also

qcsapi_gpio_monitor_reset_device

9.17.3 Function Documentation

9.17.3.1 qcsapi_gpio_set_config()

```
int qcsapi_gpio_set_config (
            const uint8_t gpio_pin,
            const qcsapi_gpio_config new_gpio_config )
```

Warning

This API can potentially damage the chip, please treat it with respect and read through the following documentation before using the API.

Configures a GPIO pin for input (1), input/output (2), or disables further use of the GPIO pin (0), as specified by the new GPIO config parameter (see qcsapi_gpio_config).

GPIO pin values run from 0 to 31.

Note

This API is only available in calibration mode (see Production mode vs calibration mode).

Parameters

gpio_pin	the GPIO to change.
new_gpio_config	the new state of the PIN.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_gpio_config <GPIO pin number> <configuration>
```

where <GPIO pin number> is a GPIO pin number, an integer in the range 0 to 31, and <configuration> is either 0, 1 or 2.

See above for the meaning of 0, 1 and 2 as a GPIO pin configuration.

Unless an error occurs, the output will be the string `complete`.

Warning

Power should not be turned off to the WiFi device when calling the set GPIO config API or immediately afterwards. Failure to follow this restriction can cause the flash memory on the board to become corrupted. If power needs to be turned off to the WiFi device when working with this API, enter the `halt` command first and wait for the device to shut down. This API should only be called when initially configuring the board.

Be aware that configuring a GPIO pin for output that either not present or wired for input can leave the board or chip open to being damaged should a set API attempt to change the GPIO pin setting to a state not supported by the hardware.

See also

qcsapi_gpio_config

9.17.3.2 qcsapi_gpio_get_config()

```
int qcsapi_gpio_get_config (
            const uint8_t gpio_pin,
            qcsapi_gpio_config * p_gpio_config )
```

Get the current configuration of a GPIO pin, either input (1), output (2), or disabled (0).

GPIO pin values are the same as in the set GPIO config API.

Parameters

gpio_pin	the GPIO to read.
p_gpio_config	return parameter to store the state of the PIN.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_gpio_config <GPIO pin number>
```

where <GPIO pin number> is a GPIO pin number, an integer in the range 0 to 31.

9.17.3.3 qcsapi_led_get()

```
int qcsapi_led_get (
            const uint8_t led_ident,
            uint8_t * p_led_setting )
```

Get the current level for an LED/GPIO pin, either HIGH (1) or LOW (0).

Note

The GPIO pin must have been previously configured for input or output thru qcsapi_gpio_set_config.

Parameters

led_ident	the GPIO pin number.
p_led_setting	

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_LED <LED / GPIO pin number>
```

where <LED / GPIO pin number> is an LED / GPIO pin number, an integer in the range 0 to 31.

Unless an error occurs, the output will be either 0 (LOW) or 1 (HIGH).

9.17.3.4 qcsapi_led_set()

```
int qcsapi_led_set (
            const uint8_t led_ident,
            const uint8_t new_led_setting )
```

Set the current level for an LED/GPIO pin, either HIGH (1) or LOW (0).

The LED identity is the GPIO pin number.

Note

The GPIO pin must have been previously configured for output thru the set GPIO config API (qcsapi_gpio_↩ set_config).

Warning

Be aware that configuring an incorrect GPIO pin for input/output and then setting the level for that invalid GPIO pin can damage the board. Consult the documentation or schematics for the board for details on the GPIO pin configuration.

Parameters

led_ident	the GPIO corresponding to the LED to change.
new_led_setting	the new state of the LED.

`call_qcsapi` interface:

```
call_qcsapi set_LED <LED / GPIO pin number> <0 | 1>
```

where <LED / GPIO pin number> is an LED / GPIO pin number, an integer in the range 0 to 31.

Unless an error occurs, the output will be the string `complete`.

Note

Most GPIO pins connect to an LED or other item of hardware that software controls directly using the get and set LED APIs. However, the WPS Push Button and the Reset Device Push Button require additional programming, since the end-user can press these push buttons to start a WPS association process or reset the WiFi device. The software thus needs to be "armed" to respond to these events. Because the way the system is expect to respond to a WPS push button press is quite different from the way it should respond to a Reset Device button press, separate APIs are provided for each.

9.17.3.5 qcsapi_led_pwm_enable()

```
int qcsapi_led_pwm_enable (
            const uint8_t led_ident,
            const uint8_t onoff,
            const qcsapi_unsigned_int high_count,
            const qcsapi_unsigned_int low_count )
```

Enable pulse wide modulation for LED GPIO pin to control LED brightness.

The LED identity is the GPIO pin number.

Parameters

led_ident	the GPIO corresponding to the LED to change.
onoff	1 to enable PWM, 0 - to disable.
high_count	'on' duration in each cycle, integer in range 1 - 256
low_count	'off' duration in each cycle, integer in range 1 - 256

`call_qcsapi` interface:

```
call_qcsapi set_LED_PWM <LED / GPIO pin number> <0 | 1> <high_count> <low↩
_count>
```

where <LED / GPIO pin number> is an LED / GPIO pin number, an integer in the range 0 to 31,

Unless an error occurs, the output will be the string `complete`.

9.17.3.6 qcsapi_led_brightness()

```
int qcsapi_led_brightness (
            const uint8_t led_ident,
            const qcsapi_unsigned_int level )
```

Set LED brightness level. Level can be beetween 1 and 10 where 10 is a maximum brightness and 1 is a lowest level before turning off the LED. The LED identity is the GPIO pin number.

Parameters

led_ident	the GPIO corresponding to the LED to change.
level	brightness level in range 1 - 10

`call_qcsapi` interface:

```
call_qcsapi set_LED_brightness <LED / GPIO pin number> <level>
```

where <LED / GPIO pin number> is an LED / GPIO pin number, an integer in the range 0 to 31,

Unless an error occurs, the output will be the string `complete`.

9.17.3.7 qcsapi_gpio_monitor_reset_device()

```
int qcsapi_gpio_monitor_reset_device (
            const uint8_t reset_device_pin,
            const uint8_t active_logic,
            const int blocking_flag,
            reset_device_callback respond_reset_device )
```

This API lets an application identify the GPIO pin connected to the reset device push button, and then monitors this push button.

Parameters

reset_device_pin	the GPIO pin that is connected to the push button. This pin must be configured for input.
active_logic	identifies whether the active state of the pin is high or low, and should be 1 or 0 respectively.
blocking_flag	specifies whether the API should block the process until the button is pressed. Currently this must be set to 1 - ie the API only supports blocking operation.
respond_reset_device	is the address of a callback entry point, with signature as per the reset_device_callback.

When called, this API (after completing error checking) periodically checks the state of reset_device_pin. When this pin goes active, as specified by active_logic, it calls the callback entry point identified by reset_device_callback. Notice the entry point is responsible for handling any response to pressing the reset device push button.

A sample requirement for how this API is used is:

- If the Reset Device Push Button is pressed for between 1 second and 5 seconds, the WiFi device reboots.

- If the Reset Device Push Button is pressed for more than 5 seconds, the factory default settings are restored and the device then reboots.

Again, the reset device callback, programming not part of the QCSAPI, is responsible for handling the response to pressing this push button.

Note

The script to restore factory default settings is expected to be located in /scripts/restore_default_config.

This API cannot be called from within `call_qcsapi`

9.17.3.8 qcsapi_gpio_enable_wps_push_button()

```
int qcsapi_gpio_enable_wps_push_button (
            const uint8_t wps_push_button,
            const uint8_t active_logic,
            const uint8_t use_interrupt_flag )
```

This API enables the WPS push button.

Unlike the reset device push button, the expected response when the WPS push button is pressed is predefined. For this reason no callback programming is required.

Parameters

wps_push_button	the GPIO used for WPS push button operation.
active_logic	identifies whether the active state of the pin is high or low, and should be 1 or 0 respectively.
use_interrupt_flag	if set to 0, selects polling operation, if 1, selects interrupt operation. If interrupt mode is selected, the active logic must be 1.

`call_qcsapi` interface:

```
call_qcsapi enable_wps_push_button <GPIO pin> <0 | 1>
```

where <GPIO pin> is the number of the GPIO pin that controls the WPS push button. The parameter that follows selects active logic, either LOW (0) or HIGH (1).

To enable the WPS push button in interrupt mode, enter: `call_qcsapi enable_wps_push_button <G↩PIO pin> 0 intr`

9.18 Per Association APIs

These APIs report on items available for each association. The first two only work on the AP; remaining APIs work on both an AP and a STA. On a STA, the association index must be 0.

Functions

- int qcsapi_wifi_get_count_associations (const char *ifname, qcsapi_unsigned_int *p_association_count)

  Get the number of STAs associated.
- int qcsapi_wifi_get_associated_device_mac_addr (const char *ifname, const qcsapi_unsigned_int device_index, qcsapi_mac_addr device_mac_addr)

  Get the associated device MAC addresses.
- int qcsapi_wifi_get_associated_device_ip_addr (const char *ifname, const qcsapi_unsigned_int device_index, unsigned int *ip_addr)

  Get the associated device IP addresses.
- int qcsapi_wifi_get_link_quality (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_link_quality)

  Get the link quality per association index.
- int qcsapi_wifi_get_link_quality_max (const char *ifname, qcsapi_unsigned_int *p_max_quality)

  Get maximum link quality for all stations.
- int qcsapi_wifi_get_rx_bytes_per_association (const char *ifname, const qcsapi_unsigned_int association_index, u_int64_t *p_rx_bytes)

  Get RX bytes per association index.
- int qcsapi_wifi_get_tx_bytes_per_association (const char *ifname, const qcsapi_unsigned_int association_index, u_int64_t *p_tx_bytes)

  Get TX bytes per association index.
- int qcsapi_wifi_get_rx_packets_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_rx_packets)

  Get RX Packets by association.
- int qcsapi_wifi_get_tx_packets_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_tx_packets)

  Get TX Packets by association.
- int qcsapi_wifi_get_tx_err_packets_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_tx_err_packets)

  Get TX Packets Errors by association.
- int qcsapi_wifi_get_rssi_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_rssi)

  Get RSSI per association index.
- int qcsapi_wifi_get_rssi_in_dbm_per_association (const char *ifname, const qcsapi_unsigned_int association_index, int *p_rssi)

  Get RSSI in dBm per association index.
- int qcsapi_wifi_get_bw_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_bw)

  Get the associated peer bandwidth (20 vs 40MHz).
- int qcsapi_wifi_get_tx_phy_rate_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_tx_phy_rate)

  Get TX PHY rate by association index.
- int qcsapi_wifi_get_rx_phy_rate_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_rx_phy_rate)

  Get RX PHY rate by association index.

- int qcsapi_wifi_get_tx_mcs_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_mcs)

  Get TX MCS by association index.
- int qcsapi_wifi_get_rx_mcs_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_mcs)

  Get RX MCS by association index.
- int qcsapi_wifi_get_achievable_tx_phy_rate_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_achievable_tx_phy_rate)

  Get Achievable TX PHY rate by association index.
- int qcsapi_wifi_get_achievable_rx_phy_rate_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_achievable_rx_phy_rate)

  Get Achievable RX PHY rate by association index.
- int qcsapi_wifi_get_auth_enc_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_auth_enc)

  Get authentification description by association index.
- int qcsapi_wifi_get_tput_caps (const char *ifname, const qcsapi_unsigned_int association_index, struct ieee8011req_sta_tput_caps *tput_caps)

  Get HT and VHT capabilities by association index.
- int qcsapi_wifi_get_connection_mode (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *connection_mode)

  Get connection mode by association index.
- int qcsapi_wifi_get_vendor_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *p_vendor)

  Get vendor by association index.
- int qcsapi_wifi_get_max_mimo (const char *ifname, const qcsapi_unsigned_int association_index, string_16 p_max_mimo)

  Get max MIMO streams by association index.
- int qcsapi_wifi_get_snr_per_association (const char *ifname, const qcsapi_unsigned_int association_index, int *p_snr)

  Get Signal to Noise Ratio (SNR) by association index.
- int qcsapi_wifi_get_time_associated_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_unsigned_int *time_associated)

  Get the associated device time per association index.
- int qcsapi_wifi_get_node_param (const char *ifname, const uint32_t node_index, qcsapi_per_assoc_param param_type, int local_remote_flag, const string_128 input_param_str, qcsapi_measure_report_result *report_result)

  Get a Parameter for a Node.
- int qcsapi_wifi_get_node_counter (const char *ifname, const uint32_t node_index, qcsapi_counter_type counter_type, int local_remote_flag, uint64_t *p_value)

  Get a Counter for a Node.
- int qcsapi_wifi_get_node_stats (const char *ifname, const uint32_t node_index, int local_remote_flag, struct qcsapi_node_stats *p_stats)

  Get the Statistics (data structure of counters) for a Node.
- int qcsapi_wifi_get_max_queued (const char *ifname, const uint32_t node_index, int local_remote_flag, int reset_flag, uint32_t *max_queued)

  Get the Maximum Number of Packets That Were Queued for the Selected Node.
- int qcsapi_wifi_get_hw_noise_per_association (const char *ifname, const qcsapi_unsigned_int association_index, int *p_hw_noise)

  Get HW Noise per association index.
- int qcsapi_wifi_get_mlme_stats_per_mac (const qcsapi_mac_addr client_mac_addr, qcsapi_mlme_stats *stats)

  Get a MLME statistics record.

- int qcsapi_wifi_get_mlme_stats_per_association (const char *ifname, const qcsapi_unsigned_int association_index, qcsapi_mlme_stats *stats)

  Get a MLME statistics record.
- int qcsapi_wifi_get_mlme_stats_macs_list (qcsapi_mlme_stats_macs *macs_list)

  Get a list of macs addresses.
- int qcsapi_wifi_sample_all_clients (const char *ifname, uint8_t *sta_count)

  this API is used to sample all client datas
- int qcsapi_wifi_get_per_assoc_data (const char *ifname, struct qcsapi_sample_assoc_data *ptr, const int num_entry, const int offset)

  this API is used to get sampled data

9.18.1 Detailed Description

9.18.2 Function Documentation

9.18.2.1 qcsapi_wifi_get_count_associations()

```
int qcsapi_wifi_get_count_associations (
            const char * ifname,
            qcsapi_unsigned_int * p_association_count )
```

Gets the number of stations currently associated with the access point. As associations are dynamic, this count can change at any time.

Note

This API is used on both AP and STA. On a STA, it is used to indicate whether it is associated with an AP.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_association_count	return parameter to store the count of STAs associated with the AP.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_count_assoc <WiFi interface>

call_qcsapi get_count_associations <WiFi interface>

call_qcsapi get_association_count <WiFi interface>
```

Unless an error occurs, the output will be the count of stations currently associated to this AP.

See also

qcsapi_wifi_get_BSSID

9.18.2.2 qcsapi_wifi_get_associated_device_mac_addr()

```
int qcsapi_wifi_get_associated_device_mac_addr (
            const char * ifname,
            const qcsapi_unsigned_int device_index,
            qcsapi_mac_addr device_mac_addr )
```

Gets the MAC address of a device (STA or WDS peer) currently associated with the AP. Second parameter selects the association index, with a range from 0 to the association count - 1. An index out of range causes the API to fail with the error set to Out of Range (-ERANGE). Use qcsapi_wifi_get_count_associations to determine the current association count.

As associations are dynamic, the count of associations can change at any time. An application should never assume that a value previously returned from qcsapi_wifi_get_count_associations remains valid. Applications should always be prepared for a return value of -ERANGE from this API, even if it just verified the number of current associations.

This API only works on an AP; for a STA, use the get BSSID API (qcsapi_wifi_get_BSSID) to get the MAC address of the associated AP.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0 or wds0).
device_index	the index of the device MAC address to return.
device_mac_addr	the MAC address of the device at index 'device_index'.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_associated_device_mac_addr <WiFi interface> <association
index>
```

The output will be the MAC address of the associated device, or an error message.

See also

qcsapi_wifi_get_BSSID

9.18.2.3 qcsapi_wifi_get_associated_device_ip_addr()

```
int qcsapi_wifi_get_associated_device_ip_addr (
            const char * ifname,
            const qcsapi_unsigned_int device_index,
            unsigned int * ip_addr )
```

Get the IP address of a device (STA or WDS peer) currently associated with the AP.

Note

This API is only used on the AP.

Second parameter selects the association index, with a range from 0 to the association count - 1. An index out of range causes the API to fail with the error set to Out of Range (-ERANGE). Use qcsapi_wifi_get_count_associations to determine the current association count. As associations are dynamic, the count of associations can change at any time. An application should never assume that a value previously returned from qcsapi_wifi_get_count_↩ associations remains valid. Applications should always be prepared for a return value of -ERANGE from this API, even if it just verified the number of current associations.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0 or wds0).
device_index	association index of STA, or 0 for a WDS peer
ip_addr	the IP address of the device at index 'device_index'.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_associated_device_ip_addr <WiFi interface> <association
index>
```

The output will be the IP address of the associated device, or an error message.

9.18.2.4 qcsapi_wifi_get_link_quality()

```
int qcsapi_wifi_get_link_quality (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_link_quality )
```

Returns the link quality as the current TX PHY rate in megabits per second (MBPS).

Note

The device must have the autorate fallback option enabled.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA link quality to be read. On the STA, this should be 0.
p_link_quality	the link quality for the given index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_link_quality <WiFi interface> <association index>
```

The output will be the current link quality for the association, in terms of the current TX PHY rate in Mbps

9.18.2.5 qcsapi_wifi_get_link_quality_max()

```
int qcsapi_wifi_get_link_quality_max (
            const char * ifname,
            qcsapi_unsigned_int * p_max_quality )
```

Returns link quality as the current TX PHY rate in megabits per second (Mbps). The function is similar to `qcsapi_wifi_get_link_quality` but returns maximum link quality for all current associations.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_max_quality	the maximum of the link quality for all current associations.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

9.18.2.6 qcsapi_wifi_get_rx_bytes_per_association()

```
int qcsapi_wifi_get_rx_bytes_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            u_int64_t * p_rx_bytes )
```

Returns the current number of bytes received on the association.

The count is set to 0 at the start of the association.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RX bytes is to be read. On the STA, this should be 0.
p_rx_bytes	return parameter to contain the number of bytes received on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rx_bytes <WiFi interface> <association index>
```

The output will be the current number of bytes received on that association.

9.18.2.7 qcsapi_wifi_get_tx_bytes_per_association()

```
int qcsapi_wifi_get_tx_bytes_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            u_int64_t * p_tx_bytes )
```

Returns the current number of bytes transmitted on the association.

The count is set to 0 at the start of the association.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA TX bytes is to be read. On the STA, this should be 0.
p_tx_bytes	return parameter to contain the number of bytes transmitted on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tx_bytes <WiFi interface> <association index>
```

The output will be the current number of bytes transmitted on that association.

9.18.2.8 qcsapi_wifi_get_rx_packets_per_association()

```
int qcsapi_wifi_get_rx_packets_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_rx_packets )
```

Returns the current number of packets received on the association.

The count is set to 0 at the start of the association.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA. On the STA, this should be 0.
p_rx_packets	return parameter to contain the number of packets received on this association.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rx_packets <WiFi interface> <association index>

call_qcsapi get_assoc_rx_packets <WiFi interface> <association index>
```

The output will be the current number of packets received on that association.

9.18.2.9 qcsapi_wifi_get_tx_packets_per_association()

```
int qcsapi_wifi_get_tx_packets_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_tx_packets )
```

Returns the current number of packets transmitted on the association.

The count is set to 0 at the start of the association.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA. On the STA, this should be 0.
p_tx_packets	return parameter to contain the number of packets transmitted on this association.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tx_packets <WiFi interface> <association index>

call_qcsapi get_assoc_tx_packets <WiFi interface> <association index>
```

The output will be the current number of packets transmitted on that association.

9.18.2.10 qcsapi_wifi_get_tx_err_packets_per_association()

```
int qcsapi_wifi_get_tx_err_packets_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_tx_err_packets )
```

Returns the current number of packets that failed to be transmitted on the association.

The count is set to 0 at the start of the association.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA. On the STA, this should be 0.
p_tx_err_packets	return parameter to contain the number of packets which failed transmission on this association.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tx_err_packets <WiFi interface> <association index>
```

The output will be the current number of packets that failed to be transmitted on the association.

9.18.2.11 qcsapi_wifi_get_rssi_per_association()

```
int qcsapi_wifi_get_rssi_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_rssi )
```

Returns the current Received Signal Strength Indication (RSSI) in the range [0, 68].

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RSSID is to be read. On the STA, this should be 0.
p_rssi	return parameter to contain the RSSI on this association index, in the range [0 - 68].

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rssi <WiFi interface> <association index>
```

The output will be the current RSSI for the association in the range [0 - 68].

9.18.2.12 qcsapi_wifi_get_rssi_in_dbm_per_association()

```
int qcsapi_wifi_get_rssi_in_dbm_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            int * p_rssi )
```

Returns the current Received Signal Strength Indication (RSSI) in dBm.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_rssi	return parameter to contain the RSSI on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rssi_dbm <WiFi interface> <association index>
```

The output will be the current RSSI in dBm for the association.

9.18.2.13 qcsapi_wifi_get_bw_per_association()

```
int qcsapi_wifi_get_bw_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_bw )
```

This API call is used to determine the bandwidth used by the peer STA. The bandwidth is 20 or 40, representing 20MHz or 40MHz.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	the entry into the association table.
p_bw	return parameter for storing the return value (either 20 or 40)

Returns

>= 0, success, < 0 error.

`call_qcsapi` interface:

```
call_qcsapi get_assoc_bw <WiFi interface> <index>
```

Unless an error occurs, the output will be one of the strings '20' or '40'.

9.18.2.14 qcsapi_wifi_get_tx_phy_rate_per_association()

```
int qcsapi_wifi_get_tx_phy_rate_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_tx_phy_rate )
```

Returns the current TX PHY rate in megabits per second (MBPS)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_tx_phy_rate	return parameter to receive the TX PHY rate on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tx_phy_rate <WiFi interface> <association index>
```

Unless an error occurs, the output will be the current TX PHY rate in MBPS.

9.18.2.15 qcsapi_wifi_get_rx_phy_rate_per_association()

```
int qcsapi_wifi_get_rx_phy_rate_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_rx_phy_rate )
```

Returns the current RX PHY rate in megabits per second (MBPS)

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_rx_phy_rate	return parameter to receive the RX PHY rate on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rx_phy_rate <WiFi interface> <association index>
```

Unless an error occurs, the output will be the current RX PHY rate in MBPS.

9.18.2.16 qcsapi_wifi_get_tx_mcs_per_association()

```
int qcsapi_wifi_get_tx_mcs_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_mcs )
```

Returns the current TX MCS

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_mcs	return parameter to receive the TX MCS on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tx_mcs <WiFi interface> <association index>
```

Unless an error occurs, the output will be the current TX MCS.

9.18.2.17 qcsapi_wifi_get_rx_mcs_per_association()

```
int qcsapi_wifi_get_rx_mcs_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_mcs )
```

Returns the current RX MCS

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_mcs	return parameter to receive the RX MCS on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_rx_mcs <WiFi interface> <association index>
```

Unless an error occurs, the output will be the current RX MCS.

9.18.2.18 qcsapi_wifi_get_achievable_tx_phy_rate_per_association()

```
int qcsapi_wifi_get_achievable_tx_phy_rate_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_achievable_tx_phy_rate )
```

Returns the achievable TX PHY rate in kilobits per second (KBPS)

Note

The units for this API are kilobits per second. The reported achievable TX PHY rate typically ranges between 54000 and 1733300.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
p_achievable_tx_phy_rate	return parameter to receive the achievable RX PHY rate on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_achievable_tx_phy_rate <WiFi interface> <association index>
```

Unless an error occurs, the output will be the achievable TX PHY rate in KBPS.

9.18.2.19 qcsapi_wifi_get_achievable_rx_phy_rate_per_association()

```
int qcsapi_wifi_get_achievable_rx_phy_rate_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_achievable_rx_phy_rate )
```

Returns the achievable RX PHY rate in kilobits per second (KBPS)

Note

The units for this API are kilobits per second. The reported achievable RX PHY rate typically ranges between 54000 and 1733300.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RSSID is to be read. On the STA, this should be 0.
p_achievable_rx_phy_rate	return parameter to receive the achievable RX PHY rate on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_achievable_rx_phy_rate <WiFi interface> <association index>
```

Unless an error occurs, the output will be the achievable RX PHY rate in KBPS.

9.18.2.20 qcsapi_wifi_get_auth_enc_per_association()

```
int qcsapi_wifi_get_auth_enc_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_auth_enc )
```

Returns the auth algorithm, key management, key protocol and cipher.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA authentification detail is to be read.
p_auth_enc	return parameter to receive the authetification details. Information is passed in packed format: Bits 0 - 7: Auth algorithm (0x00 - OPEN, 0x01 - SHARED) Bits 8 - 15: Key management (0x00 - NONE, 0x01 - EAP, 0x02 - PSK, 0x03 - WEP) Bits 16 - 23: Key protocol (0x00 - NONE, 0x01 - WPA, 0x02 - WPA2) Bits 24 - 31: Cipher (0x01 - TKIP, 0x03 - CCMP)

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_auth_enc_per_assoc <WiFi interface> <association index>
```

Unless an error occurs, the output will be the authenification details.

9.18.2.21 qcsapi_wifi_get_tput_caps()

```
int qcsapi_wifi_get_tput_caps (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            struct ieee8011req_sta_tput_caps * tput_caps )
```

Returns the contents of the HT and VHT information elements for an associated station or access point.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	the association index of an associated station (0 for AP)
tput_caps	buffer to receive the returned data

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_tput_caps <WiFi interface> <association index>
```

Unless an error occurs, the output will be the content of the station's HT and VHT information elements in hex format. LSB to MSB order will be used for output.

9.18.2.22 qcsapi_wifi_get_connection_mode()

```
int qcsapi_wifi_get_connection_mode (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * connection_mode )
```

Returns the connection mode for an associated station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	the association index of an associated station
connection_mode	buffer to receive the current mode, which is a value from the `ieee80211_wifi_modes` enum

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_connection_mode <WiFi interface> <association index>
```

Unless an error occurs, the output will be a WiFi mode from the `qcsapi_wifi_modes_strings` array. E.g. 'a', 'b', 'g', 'na', 'ng' or 'ac'.

9.18.2.23 qcsapi_wifi_get_vendor_per_association()

```
int qcsapi_wifi_get_vendor_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * p_vendor )
```

Returns vendor name of the peer device (if known) for the given association index.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RSSID is to be read. On the STA, this should be 0.
p_vendor	return vendor name for this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_vendor <WiFi interface> <association index>
```

Unless an error occurs, the output will be the vendor of the client.

9.18.2.24 qcsapi_wifi_get_max_mimo()

```
int qcsapi_wifi_get_max_mimo (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            string_16 p_max_mimo )
```

Returns the maximum number of receive and transmit spatial streams allowed to/from an associated station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	the association index of an associated station
p_max_mimo	buffer to receive the max MIMO streams

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_max_mimo <WiFi interface> <association index>
```

Unless an error occurs, the output will be a string representing max MIMO streams supported by the station in the form "Rx:A Tx:B" where A and B are integers. The output will be "unknown" if the number of streams cannot be determined.

9.18.2.25 qcsapi_wifi_get_snr_per_association()

```
int qcsapi_wifi_get_snr_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            int * p_snr )
```

Returns the current SNR for the given association index.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RSSID is to be read. On the STA, this should be 0.
p_snr	return parameter to receive the SNR on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_snr <WiFi interface> <association index>
```

Unless an error occurs, the output will be the current SNR.

9.18.2.26 qcsapi_wifi_get_time_associated_per_association()

```
int qcsapi_wifi_get_time_associated_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_unsigned_int * time_associated )
```

Returns the time in seconds a STA has been associated with an AP. This API can be applied to both station and AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA to be read. On the STA, this should be 0.
time_associated	return parameter to contain the time associated on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_time_associated <WiFi interface> 0
```

The output will be the time in seconds that the STA has been associated with an AP.

For the AP, the final parameter is

```
call_qcsapi get_time_associated <WiFi interface> <association index>
```

The output will be the time in seconds that the STA at association index has been associated with an AP.

9.18.2.27 qcsapi_wifi_get_node_param()

```
int qcsapi_wifi_get_node_param (
            const char * ifname,
            const uint32_t node_index,
            qcsapi_per_assoc_param param_type,
            int local_remote_flag,
            const string_128 input_param_str,
            qcsapi_measure_report_result * report_result )
```

Returns the value of the selected parameter for the selected node.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
node_index	selects the node to report.
param_type	the parameter type. See the definition of the enun `qcsapi_per_assoc_param`.
local_remote_flag	use local flag to get local parameters, use remote flag to get parameters from remote associated STA; set to `QCSAPI_LOCAL_NODE` or `QCSAPI_REMOTE_NODE`
input_param_str	address to related request parameters, actual request structure information please refer to qcsapi_measure_request_param
report_result	address to receive all kinds of results

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_node_param <WiFi interface> <node index> <parameter name>
<local remote flag> <paramteres indicated by type>
```

where `parameter name` is one of `link_quality`, `tx_phy_rate`, `rx_phy_rate`, `rssi_dbm`, `snr`, `rssi`, `hw_noise`, `soc_macaddr`, `bw`, `basic`, `cca`, `rpi`, `chan_load`, `noise_histogram`, `beacon`

The output will be the value for the selected parameter. RSSI_DBM will be in dBm; SNR and RSSI will be in dB.

9.18.2.28 qcsapi_wifi_get_node_counter()

```
int qcsapi_wifi_get_node_counter (
            const char * ifname,
            const uint32_t node_index,
            qcsapi_counter_type counter_type,
            int local_remote_flag,
            uint64_t * p_value )
```

Returns the value of the selected counter for the selected node.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
node_index	selects the node to report.
counter_type	the counter to select. See the definition of the enun `qcsapi_counter_type`.
local_remote_flag	use local flag to get local counters, use remote flag to get counters from remote associated STA; set to `QCSAPI_LOCAL_NODE` or `QCSAPI_REMOTE_NODE`
p_value	address to receive the value of the parameter. It must address a 64-bit quantity.

Note

Not all per-node counters are 64 bits wide. Some will roll over when the maximum 32-bit value is reached.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_node_counter <WiFi interface> <node index> <counter type>
<local remote flag>
```

where `counter name` is as defined in the section on the Get Counter API.

The output will be the value for the selected counter.

9.18.2.29 qcsapi_wifi_get_node_stats()

```
int qcsapi_wifi_get_node_stats (
            const char * ifname,
            const uint32_t node_index,
            int local_remote_flag,
            struct qcsapi_node_stats * p_stats )
```

Returns a data structure populated with statistics (counters) for a node.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
node_index	selects the node to report.
local_remote_flag	use local flag to get local statistics, use remote flag to get statistics from remote associated STA; set to `QCSAPI_LOCAL_NODE` or `QCSAPI_REMOTE_NODE`
p_stats	address of a `struct qcsapi_node_stats`

Note

See the definition of the Node Stats data struct for details on what counters this API will return.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_node_stats <WiFi interface> <node index> <local remote
flag>
```

The output will be the contents of the Node Stats data struct, one value per line. The name of the field is also displayed.

9.18.2.30 qcsapi_wifi_get_max_queued()

```
int qcsapi_wifi_get_max_queued (
            const char * ifname,
            const uint32_t node_index,
            int local_remote_flag,
            int reset_flag,
            uint32_t * max_queued )
```

Returns the maxiumum number of packets that were queued for the selected node.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
node_index	selects the node to report.
local_remote_flag	use local flag to get local max queued packets, use remote flag to get max queues packets from remote associated STA; set to QCSAPI_LOCAL_NODE or QCSAPI_REMOTE_NODE
reset_flag	whether to reset the statistic on read. "1" to reset and "0" not to reset.
max_queued	address of a 32-bit unsigned integer to receive the value

`call_qcsapi` interface:

```
call_qcsapi get_max_queued <WiFi interface> <node index> <local remote
flag> <reset flag>
```

The output will be the maximum number of packets that were queued for the selected node index.

9.18.2.31 qcsapi_wifi_get_hw_noise_per_association()

```
int qcsapi_wifi_get_hw_noise_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            int * p_hw_noise )
```

Returns the current HW noise.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
association_index	On the AP this is the association index of the STA RSSID is to be read. On the STA, this should be 0.
p_hw_noise	return parameter to contain the hw_noise on this association index.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_hw_noise <WiFi interface> <association index>
```

The output will be the current HW noise for the association in the range.

9.18.2.32 qcsapi_wifi_get_mlme_stats_per_mac()

```
int qcsapi_wifi_get_mlme_stats_per_mac (
            const qcsapi_mac_addr client_mac_addr,
            qcsapi_mlme_stats * stats )
```

This API returns the mlme statistics record for specified mac address.

Parameters

client_mac_addr	the mac addr of the client. 00:00:00:00:00:00 should be used to get mlme stats for clients who were not associated with an AP.
stats	address of a `struct qcsapi_mlme_stats`

Returns

0 if the stats record for required mac address exists.

-qcsapi_mlme_stats_not_supported if statistics facility is not suported.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mlme_stats_per_mac <mac_addr>
```

Unless an error occurs, the output will be the mlme statistics for specified mac.

9.18.2.33 qcsapi_wifi_get_mlme_stats_per_association()

```
int qcsapi_wifi_get_mlme_stats_per_association (
            const char * ifname,
            const qcsapi_unsigned_int association_index,
            qcsapi_mlme_stats * stats )
```

This API returns the mlme statistics record for specified association index.

Parameters

ifname	the interface to perform the action on.
association_index	the association index to get mlme statistics about
stats	address of a `struct qcsapi_mlme_stats`

Note

This API is only available on AP mode interface.

Returns

0 if the stats record for required association index address exists.

-qcsapi_mlme_stats_not_supported if statistics facility is not suported.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mlme_stats_per_association <WIFI interface> <association
index>
```

Unless an error occurs, the output will be the mlme statistics for specified association index.

9.18.2.34 qcsapi_wifi_get_mlme_stats_macs_list()

```
int qcsapi_wifi_get_mlme_stats_macs_list (
            qcsapi_mlme_stats_macs * macs_list )
```

This API returns the list of macs currently existing in mlme statistic factory.

Parameters

macs_list	address of a `struct qcsapi_mlme_stats_macs`

Returns

0 if the list obtained successfully.

-qcsapi_mlme_stats_not_supported if statistics facility is not suported.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mlme_stats_macs_list
```

Unless an error occurs, the output will be the list of the mac addresses existing in mlme statistics table.

9.18.2.35 qcsapi_wifi_sample_all_clients()

```
int qcsapi_wifi_sample_all_clients (
            const char * ifname,
            uint8_t * sta_count )
```

To sample all conected station records

Parameters

ifname	wireless interface name
sta_count	pointer for number of clients

Returns

>=0 on success, <0 on error.

`call_qcsapi` interface:

```
call_qcsapi sample_all_clients <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.18.2.36 qcsapi_wifi_get_per_assoc_data()

```
int qcsapi_wifi_get_per_assoc_data (
            const char * ifname,
            struct qcsapi_sample_assoc_data * ptr,
            const int num_entry,
            const int offset )
```

To get the record of conected station

Parameters

ifname	wireless interface name
ptr	client records pointer
num_entry	station count
offset	station offset

Returns

>=0 on success, <0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_assoc_data <WiFi interface> <num entry> <offset>
```

Unless an error occurs, the output will be the string `complete`.

9.19 APIs for Regulatory Compliance

This section describes APIs that assist with regulatory compliance. "Get" APIs are present for reference and convenience; "set" APIs are recommended to insure regulatory compliance. Also included are details on how to configure and manage transmit (TX) power, since the "set" APIs rely on this TX power configuration to set the transmit power. All APIs qcsapi_regulatory_xxx are used for regulatory database, and other APIs qcsapi_wifi_xxx which are used for EIRP table will be discarded.

Functions

- int qcsapi_wifi_get_list_regulatory_regions (string_256 list_regulatory_regions)

  Get the list of WiFi regulatory regions.
- int qcsapi_regulatory_get_list_regulatory_regions (string_256 list_regulatory_regions)

  Get the list of WiFi regulatory regions from regulatory database.
- int qcsapi_wifi_get_list_regulatory_channels (const char *region_by_name, const qcsapi_unsigned_int bw, string_1024 list_of_channels)

  Get the List of Regulatory Channels.
- int qcsapi_regulatory_get_list_regulatory_channels (const char *region_by_name, const qcsapi_unsigned_int bw, string_1024 list_of_channels)

  Get the List of Regulatory Channels from regulatory database.
- int qcsapi_regulatory_get_list_regulatory_bands (const char *region_by_name, string_128 list_of_bands)

  Get the List of Regulatory bands from regulatory database.
- int qcsapi_wifi_get_regulatory_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, int *p_tx_power)

  Get the Regulatory Transmit Power.
- int qcsapi_regulatory_get_regulatory_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, int *p_tx_power)

  Get the Regulatory Transmit Power from regulatory database.
- int qcsapi_wifi_get_configured_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, const qcsapi_unsigned_int bw, int *p_tx_power)

  Get WiFi Configured TX power.
- int qcsapi_regulatory_get_configured_tx_power (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, const qcsapi_unsigned_int bw, int *p_tx_power)

  Get WiFi Configured TX power from regulatory database.
- int qcsapi_regulatory_get_configured_tx_power_ext (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, const qcsapi_bw the_bw, const qcsapi_unsigned_int bf_on, const qcsapi_unsigned_int number_ss, int *p_tx_power)

  Get WiFi Configured TX power from regulatory database.
- int qcsapi_wifi_set_regulatory_region (const char *ifname, const char *region_by_name)

  Set the Regulatory Region.
- int qcsapi_regulatory_set_regulatory_region (const char *ifname, const char *region_by_name)

  Set the Regulatory Region supported by regulatory database.
- int qcsapi_regulatory_restore_regulatory_tx_power (const char *ifname)

  restore TX power by regulatory database
- int qcsapi_wifi_get_regulatory_region (const char *ifname, char *region_by_name)

  Get the current Regulatory Region.
- int qcsapi_regulatory_overwrite_country_code (const char *ifname, const char *curr_country_name, const char *new_country_name)

  Overwrite country code, mainly set speicfic country string in country IE when EU region is used.
- int qcsapi_wifi_set_regulatory_channel (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, const qcsapi_unsigned_int tx_power_offset)

Set WiFi Regulatory Channel.

- int qcsapi_regulatory_set_regulatory_channel (const char *ifname, const qcsapi_unsigned_int the_channel, const char *region_by_name, const qcsapi_unsigned_int tx_power_offset)

  Set WiFi Regulatory Channel supported by regulatory database.

- int qcsapi_regulatory_get_db_version (int *p_version, const int index)

  Get regulatory database version number from regulatory database.

- int qcsapi_regulatory_apply_tx_power_cap (int capped)

  set if tx power is capped by regulatory database.

9.19.1 Detailed Description

9.19.2 Overview

Regulatory compliance covers the choice of WiFi channels and how much power can be transmitted on each channel. The choice of WiFi channels is pretty straightforward. Typically either a channel is available or is not available. Currently the APIs either grant access to a channel or block access to that channel (if a channel cannot be used the API returns -EINVAL as its error code return value). If a channel is available by regulatory authority, no further restrictions are imposed on its use, except for the amount of power that can be transmitted.

Transmit power (TX power) is more complicated. First, the regulatory authority imposes a limit on the overall TX power. But sometimes the TX power that should be configured is lower than the limit established by the regulatory authority. For example, each board design usually has a maximum TX power for the board, a power level that should not be exceeded on any channel. Also, during testing for regulatory compliance, sometimes it is found the TX power for a particular channel needs to be reduced to meet the detailed requirements of the regulatory authority. This latter limit on the TX power can change from channel to channel, and from one regulatory region to another.

Transmit power is always measured and reported for an individual antenna chain. The overall TX power that the system broadcasts is not measured or considered. It is expect regulatory requirements will be mapped to TX power limitations on a per-chain / per-antenna basis, and that testing for regulatory compliance will be based on per-chain / per-antenna results.

9.19.3 Managing TX Power

The TX power that should be configured for a channel is derived from the three sources described above:

1. TX power limit set by regulatory authority.
2. TX power limit established for the board. This limit is independent of the regulatory authority, but can lower the TX power below what the regulatory authority specifies.
3. TX power limit established for each channel - the TX power database. The limit for each channel can differ depending on the controlling regulatory authority. This limit also can lower the TX power below what the regulatory authority specifies.

All TX power values are expressed as integers in units of dBm. All values are absolute power settings, not offsets. The rule for deriving the TX power for a channel is: take the minimum of the values from each of the three sources, regulatory authority, board limit and the TX power database.

9.19.4 The Calibration State

A boot configuration environmental variable, calstate, helps determine the capabilities of the WiFi device when it boots up. If calstate is set to 1, the system installs special programming that facilitates testing for regulatory compliance. In this state though, the WiFi device will not associate or pass traffic. Set calstate to 3 to get the system to boot up so that it will associate and pass traffic. With calstate set to 3 though, testing for regulatory compliance may be less straightforward. This latter state is also referred to as production mode.

To make changes in the TX power database, calstate must be set to 1.

9.19.5 Selecting the Regulatory Region

The regulatory region is specified using a string. The table below lists currently supported regulatory regions and what strings will select that region. Base name is what would be returned by the Get Regulatory Region and the Get List of WiFi Regulatory Region APIs.

Region	Base Name	Synonyms

United States of America – Federal Communications Commission (FCC)

us

US, usa, USA, FCC

European Community

eu

EU, CE, ce, Europe

9.19.6 TX Power Configuration

This section describes where the 3 sources for configuring TX power are located.

9.19.7 Limit by Regulatory Authority

The TX power limit that is set by regulatory authority is stored in tables that are part of the binary QCSAPI library (libqcsapi.so) and cannot be edited. The value for each available channel is available from an API. The same API also reports if a channel is available, for if it is called with an invalid channel, it will return an error (error code return value: `-EINVAL`). An example is WiFi channel 188, a valid 5 GHz channel by the 802.11 standard, but not available in Europe or the USA.

9.19.8 Per Channel Limits

The TX power limit established for each channel is stored in flash memory, in the boot configuration sector. (All boards are required to have a boot configuration sector in flash memory.) Entries in this table are typically obtained while testing the WiFi device at a laboratory that verifies regulatory compliance. Here is an example file of TX power limits configured for each WiFi channel:

```
# TX power database table
# Chan    TX power
  36       14
  40       15
  44       15
  48       15
  52       15
  56       15
  60       15
  64       14
  100      18
  104      19
  108      19
  112      19
  116      19
  132      19
  136      19
  140      18
```

(Caution: above table is an example and should not be used. Actual values for the max TX power have to be derived from testing for regulatory compliance.)

Each channel is listed together with the max TX power for that channel. The table should then be stored in a file, with one entry per line. Each line is expected to have 2 numbers (3 numbers are also possible, as explained later). The first number is the channel; the second is the max TX power for that channel. A separate file should be created for each regulatory region.

This file then needs to be loaded into the TX power database, as shown in section Reviewing and Updating the TX Power Configuration

9.19.9 Board Limits

The overall max TX power for the board is stored in a boot configuration environmental variable, `max_tx_power`. A related variable, `min_tx_power`, stores the minimum TX power that should be configured. If the regulatory limit for transmit power for a a channel is below the value for `min_tx_power`, access to that channel will be denied under that regulatory region.

Values for both are expected to be obtained from testing the board.

9.19.10 Bandwidth 20 MHz vs. Bandwidth 40 MHz

The usual bandwidth when the system is in production mode is 40 MHz. However, it is possible to configure the bandwidth to be 20 MHz. When configured this way, the WiFi device will limit the bandwidth to 20 MHz, even if its partner in association supports 40 MHz.

It turns out the per-channel power limits can differ based on the configured bandwidth, 20 MHz vs 40 MHz. Regulatory requirements though do not change based on how the bandwidth is configured; nor does the overall board limit, the value in the boot configuration environmental variable, `max_tx_power`.

To support a different TX power limit when the bandwidth is configured to 20 MHz, each entry in the TX power database table can have 2 TX power values. The first value is the one to be used if the bandwidth is configured to 40 MHz. If present, a second value will be used if the bandwidth is configured to 20 MHz. This second value is optional. If this second value is not present, the same TX power will be used for bandwidth configured to 40 MHz or 20 MHz, obtained from the first entry.

Below is an example file of TX power limits with separate entries for bandwidth of 40 MHz and 20 MHz.

```
# TX power database table
# chan 40/20  20
#-----------------
   36   14    14
   40   14    15
   44   15    15
   48   15    15
   52   15    15
   56   15    15
   60   14    15
   64   14    15
   100   15    15
   104   15    17
   108   17    17
   112   17    17
   116   17    17
   132   17    17
   136   15    17
   140   15    15
```

(Caution: above table is an example and should not be used. Actual values for the max TX power need to be derived from testing for regulatory compliance.)

9.19.11 Defaults for TX Power

Regulatory limits are stored in the QCSAPI library binary and cannot be changed. The per-channel and board limits are expected to be configured for each board. If either are missing, the API programming will create defaults.

For the boot configuration environmental variables, the default value for `max_tx_power` is 19; for `min_tx_power` it is 9.

The TX power database has a separate table for each regulatory region. The table has an entry of channel, max TX power for each channel. If this table is absent for a regulatory region, the software will create this table, using the minimum of the regulatory limit and the board limit to obtain the max TX power for each channel. Note the default board limit is 19 dBm, so the "default default" max TX power is 19 dBm. Only one TX power value will be configured for each channel, so by default no distinction will be made between bandwidth configured to 40 MHz vs 20 MHz.

These defaults are provided to prevent problems if for any reason the per-channel limits or the board limits are not configured. It is expected that both will be configured before bringing the system up in production mode (`calstate = 3`), using the commands described in the next section.

9.19.12 Reducing TX Power on the STA Independent of the AP

For various reasons it may be necessary to reduce power on the STA separately from the AP. This distinction is required because a particular WiFi device can switch roles, Access Point and Station, by simply changing the configuration. If a device changes from STA mode to AP mode, any power reduction applied due to it being a STA should no longer be applied.

For this reason, another boot configuration parameter, `max_sta_tx_power`, is available that will limit the max TX power, but only on the STA. This parameter is optional; If not present, the system will look for and work with `max_tx_power` as described previously.

9.19.13 Reviewing and Updating the TX Power Configuration

As mentioned previously, regulatory limited are stored in the API binaries and cannot be modified. Use the API Get Regulatory TX Power, described in detail below, to access regulatory requirements, including the maximum TX power allowed by regulatory authority. Use the API Get List Regulatory Channels, described in detail below, to display the list of channels allowed in a particular regulatory region.

The boot configuration environmental variables `calstate, max_tx_power` and `min_tx_power` can all be accessed from the u-boot prompt, or by using the `get_bootval` command once the system boots up. Their values can be changed at the u-boot prompt, or by using the `set_bootval` command after the system boots up. Be aware that when setting a value from the u-boot prompt, the new value needs to be saved with the `saveenv` u-boot command. Updates to boot configuration environmental variables are automatically saved with the `set_↩ bootval` command.

Three commands are available to work with the TX power database, the tables of TX power limits for each channel. The boot parameter `calstate` must be set to 1 to make changes to the TX power database.

The command `configure_tx_power_limit` sets up the TX power database for a particular regulatory region. The command takes either one or two parameters. If two parameters are present, the first is interpreted as the regulatory region and the second as the path to a file with the table of channels and TX power limits. Example file contents are shown in sections 8.15.7 and 8.15.9. If only one parameter is present, it will be interpreted as the regulatory region. See section Selecting the Regulatory Region for details on selecting a regulatory region. Boot parameter `calstate` must be set to 1.

The command `display_tx_power_limit` displays the TX power limits for a particular channel, or the entire table for a regulatory region. This command expects two parameters. The first is the regulatory region; the second is the WiFi channel. If the 2nd parameter is "all", the entire table for the selected regulatory region is displayed. See section HERE for details on selecting a regulatory region.

The command `update_tx_power_limit` updates the TX power limits for a particular channel. This command takes either three or four parameters. The first is the regulatory region; the second is the WiFi channel; the third is the limit on TX power. A fourth parameter will be interpreted as the limit on TX power when the bandwidth is configured to 20 MHz. Boot parameter `calstate` must be set to 1.

9.19.14 Function Documentation

9.19.14.1 qcsapi_wifi_get_list_regulatory_regions()

```
int qcsapi_wifi_get_list_regulatory_regions (
            string_256 list_regulatory_regions )
```

Use this API to get a list of the regulatory regions supported by the current firmware. String will contain a list of regions, each separated by a comma.

Parameters

list_regulatory_regions	the string where the results are returned.

Returns

-EFAULT on error, or 0.

`call_qcsapi` interface:

```
call_qcsapi get_list_regulatory_regions <WiFi interface>
```

The output will be the list of regulatory regions that the firmware supports. Some listed regions may be synonyms, e.g. "Europe" and "eu" are synonyms as are "USA" and "us".

9.19.14.2 qcsapi_regulatory_get_list_regulatory_regions()

```
int qcsapi_regulatory_get_list_regulatory_regions (
            string_256 list_regulatory_regions )
```

Use this API to get a list of the regulatory regions supported by the current firmware. String will contain a list of regions, each separated by a comma.

Parameters

list_regulatory_regions	the string where the results are returned.

Returns

-EFAULT on error, or 0.

`call_qcsapi` interface:

```
call_qcsapi get_list_regulatory_regions <WiFi interface>
```

The output will be the list of regulatory regions that the firmware supports. Some listed regions may be synonyms, e.g. "Europe" and "eu" are synonyms as are "USA" and "us".

9.19.14.3 qcsapi_wifi_get_list_regulatory_channels()

```
int qcsapi_wifi_get_list_regulatory_channels (
            const char * region_by_name,
            const qcsapi_unsigned_int bw,
            string_1024 list_of_channels )
```

Use this API to get the list of channels authorized for use in the indicated regulatory region. Bandwidth parameter should be either 20 or 40. Valid channels are returned in the `list_of_channels` parameter as a list of numeric values separated by commas. This API is provided as a reference and a convenience; its use is not required to insure regulatory compliance.

Parameters

region_by_name	the regulatory region for which the channel list is expected.
bw	the bandwidth that is currently used. 40Mhz or 20Mhz.
list_of_channels	the list of channels returned.

Returns

-EFAULT, -EINVAL, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_channels <region> <20 | 40>

call_qcsapi get_list_regulatory_channels <region> <20 | 40>
```

where `<regulatory region>` should be one of the regions listed in by the get list regulatory regions API / command. Final parameter is the bandwidth and is optional. If not present, the system will use the current configured bandwidth, defaulting to 40 if that cannot be established. Output is the list of channels valid for that region separated by commas.

Example:
`call_qcsapi get_list_regulatory_channels eu`

9.19.14.4 qcsapi_regulatory_get_list_regulatory_channels()

```
int qcsapi_regulatory_get_list_regulatory_channels (
            const char * region_by_name,
            const qcsapi_unsigned_int bw,
            string_1024 list_of_channels )
```

Use this API to get the list of channels authorized for use in the indicated regulatory region. Bandwidth parameter should be 20, 40 or 80. Valid channels are returned in the `list_of_channels` parameter as a list of numeric values separated by commas. This API is provided as a reference and a convenience; its use is not required to insure regulatory compliance.

Parameters

region_by_name	the regulatory region for which the channel list is expected.
bw	the bandwidth that is currently used. 80MHz, 40Mhz or 20Mhz.
list_of_channels	the list of channels returned.

Returns

-EFAULT, -EINVAL, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_channels <region> <20 | 40 | 80>

call_qcsapi get_list_regulatory_channels <region> <20 | 40 | 80>
```

where `<regulatory region>` should be one of the regions listed in by the get list regulatory regions API / command. Final parameter is the bandwidth and is optional. If not present, the system will use the current configured bandwidth, defaulting to 80 if that cannot be established. Output is the list of channels valid for that region separated by commas.

Example:
`call_qcsapi get_list_regulatory_channels eu`

9.19.14.5 qcsapi_regulatory_get_list_regulatory_bands()

```
int qcsapi_regulatory_get_list_regulatory_bands (
            const char * region_by_name,
            string_128 list_of_bands )
```

Use this API to get the list of band authorized for use in the indicated regulatory region. Valid channels are returned in the `list_of_bands` parameter as a list of numeric values separated by commas. This API is provided as a reference and a convenience; its use is not required to insure regulatory compliance.

Parameters

region_by_name	the regulatory region for which the channel list is expected.
list_of_bands	the list of bands returned.

Returns

-EFAULT, -EINVAL, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_list_regulatory_bands <regulatory region>

where <regulatory region> should be one of the regions listed in by the get
list regulatory regions API / command.  Output is the list of bands valid
for that region separated by commas.

Example:
call_qcsapi get_list_regulatory_bands eu
```

9.19.14.6 qcsapi_wifi_get_regulatory_tx_power()

```
int qcsapi_wifi_get_regulatory_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            int * p_tx_power )
```

This API call gets the transmit power in a regulatory region for a particular channel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is returned.
p_tx_power	the result which contains the transmit power.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_tx_power <WiFi interface> <channel> <region>
```

Unless an error occurs, the output will be the channel number. eg. 20
Examples:
A valid call:

```
quantenna # call_qcsapi get_regulatory_tx_power wifi0 100 eu
22
```

An invalid call:

```
quantenna # call_qcsapi get_regulatory_tx_power wifi0 188 eu
QCS API error 22: Invalid argument
```

9.19.14.7 qcsapi_regulatory_get_regulatory_tx_power()

```
int qcsapi_regulatory_get_regulatory_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            int * p_tx_power )
```

This API call gets the transmit power in a regulatory region for a particular channel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is returned.
p_tx_power	the result which contains the transmit power.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_tx_power <WiFi interface> <channel> <region>
```

Unless an error occurs, the output will be the channel number. eg. 20
Examples:
A valid call:

```
quantenna # call_qcsapi get_regulatory_tx_power wifi0 100 eu
22
```

An invalid call:

```
quantenna # call_qcsapi get_regulatory_tx_power wifi0 188 eu
QCS API error 22: Invalid argument
```

9.19.14.8 qcsapi_wifi_get_configured_tx_power()

```
int qcsapi_wifi_get_configured_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            const qcsapi_unsigned_int bw,
            int * p_tx_power )
```

This API call gets the configured transmit power in a regulatory region for a particular channel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is returned.
bw	the bandwidth that is currently used. 40Mhz or 20Mhz.
p_tx_power	the result which contains the transmit power.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_configured_tx_power <WiFi interface> <channel> <region>
```

Unless an error occurs, the output will be the channel number. Examples:

```
quantenna # call_qcsapi get_configured_tx_power wifi0 100 eu 40
19
quantenna # call_qcsapi get_configured_tx_power wifi0 100 eu 20
19
quantenna # call_qcsapi get_configured_tx_power wifi0 64 eu 40
15
quantenna # call_qcsapi get_configured_tx_power wifi0 64 eu 20
15
quantenna # call_qcsapi get_configured_tx_power wifi0 188 eu 20
QCSAPI error 22: Invalid argument
```

Note: Numeric TX power results are just examples. Actual TX Power values may differ from what is shown above.

9.19.14.9 qcsapi_regulatory_get_configured_tx_power()

```
int qcsapi_regulatory_get_configured_tx_power (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            const qcsapi_unsigned_int bw,
            int * p_tx_power )
```

This API call gets the configured transmit power in a regulatory region for a particular channel, for one spatial stream and beamforming off. Please use qcsapi_regulatory_get_configured_tx_power_ext() to obtain maximum allowed TX power taking into consideration beamforming and number of spatial streams.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is returned.
bw	the bandwidth that is currently used. 80Mhz, 40Mhz or 20Mhz.
p_tx_power	the result which contains the transmit power.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_configured_tx_power <interface> <channel> <region> <bandwidth>
```

Unless an error occurs, the output will be the channel number. Examples:

```
quantenna # call_qcsapi get_configured_tx_power wifi0 100 eu 40
19
quantenna # call_qcsapi get_configured_tx_power wifi0 100 eu 20
19
quantenna # call_qcsapi get_configured_tx_power wifi0 64 eu 40
15
quantenna # call_qcsapi get_configured_tx_power wifi0 64 eu 20
15
quantenna # call_qcsapi get_configured_tx_power wifi0 188 eu 20
QCSAPI error 22: Invalid argument
```

Note

This API is deprecated and replaced with qcsapi_regulatory_get_configured_tx_power_ext.

9.19.14.10 qcsapi_regulatory_get_configured_tx_power_ext()

```
int qcsapi_regulatory_get_configured_tx_power_ext (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            const qcsapi_bw the_bw,
            const qcsapi_unsigned_int bf_on,
            const qcsapi_unsigned_int number_ss,
            int * p_tx_power )
```

This API call gets the configured transmit power in a regulatory region for a particular channel and number of spatial streams.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region.
the_bw	the bandwidth that is currently used. 80Mhz, 40Mhz or 20Mhz.
bf_on	beamforming is either on or off. 1 for beamforming on and 0 for beamforming off.
number_ss	the number of spatial streams.
p_tx_power	the result which contains the transmit power.

Returns

-EFAULT or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_configured_tx_power <interface> <channel> <region> <bandwidth>
<bf_on> <num_ss>
```

Unless an error occurs, the output will be the channel number. Examples:

```
quantenna # call_qcsapi get_configured_tx_power wifi0 100 us 80 1 4
15
quantenna # call_qcsapi get_configured_tx_power wifi0 100 us 20 0 2
17
```

Note: Numeric TX power results are just examples. Actual TX Power values may differ from what is shown above.

9.19.14.11 qcsapi_wifi_set_regulatory_region()

```
int qcsapi_wifi_set_regulatory_region (
            const char * ifname,
            const char * region_by_name )
```

This API call sets the regulatory region on a given interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
region_by_name	the regulatory region.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_regulatory_region <WiFi interface> <region>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.12 qcsapi_regulatory_set_regulatory_region()

```
int qcsapi_regulatory_set_regulatory_region (
            const char * ifname,
            const char * region_by_name )
```

This API call sets the regulatory region on a given interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
region_by_name	the regulatory region.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_regulatory_region <WiFi interface> <region>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.13 qcsapi_regulatory_restore_regulatory_tx_power()

```
int qcsapi_regulatory_restore_regulatory_tx_power (
            const char * ifname )
```

This API call restore TX power by regulatory database

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi restore_regulatory_tx_power <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.14 qcsapi_wifi_get_regulatory_region()

```
int qcsapi_wifi_get_regulatory_region (
            const char * ifname,
            char * region_by_name )
```

This API call gets the current regulatory region on a given interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
region_by_name	the regulatory region that is currently configured.

Returns

-EFAULT or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_region <WiFi interface>
```

Unless an error occurs, the output will be the '<region>'.

9.19.14.15 qcsapi_regulatory_overwrite_country_code()

```
int qcsapi_regulatory_overwrite_country_code (
            const char * ifname,
            const char * curr_country_name,
            const char * new_country_name )
```

This API call sets secific country code for EU region on a given interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
curr_country_name	the current country name.
new_country_name	the specific country name.

Returns

-EFAULT, -EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi overwrite_country_code <WiFi interface> <curr_country_name>
<new_country_name>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.16 qcsapi_wifi_set_regulatory_channel()

```
int qcsapi_wifi_set_regulatory_channel (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            const qcsapi_unsigned_int tx_power_offset )
```

This API call sets the transmit power adjusting the offset on a given channel on a given region(should be current region) for the passed in interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is modified
tx_power_offset	the offset in integer from the currently configured tx power.

Returns

-EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_regulatory_channel <WiFi interface> <channel> <region>
<offset>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.17 qcsapi_regulatory_set_regulatory_channel()

```
int qcsapi_regulatory_set_regulatory_channel (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            const char * region_by_name,
            const qcsapi_unsigned_int tx_power_offset )
```

This API call sets the transmit power adjusting the offset on a given channel on a given region(should be current region) for the passed in interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
the_channel	the channel for which the tx power is returned.
region_by_name	the regulatory region for which the tx power is modified
tx_power_offset	the offset in integer from the currently configured tx power.

Returns

-EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_regulatory_channel <WiFi interface> <channel> <region>
<offset>
```

Unless an error occurs, the output will be the string `complete`.

9.19.14.18 qcsapi_regulatory_get_db_version()

```
int qcsapi_regulatory_get_db_version (
            int * p_version,
            const int index )
```

This API call gets the regulatory database version number

Parameters

p_version	pointer to save version number
index	- which version number will be retrieved

Returns

-EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_regulatory_db_version [index]
```

Unless an error occurs, the output will be the version number.

9.19.14.19 qcsapi_regulatory_apply_tx_power_cap()

```
int qcsapi_regulatory_apply_tx_power_cap (
            int capped )
```

This API call set TX power capped by regulatory database

Parameters

capped	- zero for no capped by databse, non-zero for capped by database

Returns

-EOPNOTSUPP or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi apply_regulatory_cap <interface> <0|1>
```

Unless an error occurs, the output will be the string `complete`.

9.20 DFS, Radar and OCAC APIs

Selected channels in the 5 GHz frequency range are also used by weather and military radar. To prevent unwanted interference, WiFi devices using those channels are required to follow special protocols, as describe in 802.11h. APIs are available that list channels that are subject to the DFS protocols and channels that are not subject to those protocols. A separate API reports whether a particular channel is subject to DFS protocols. As the exact DFS-related regulations are determined by the regulatory authority, the regulatory region is a required parameter for all these APIs.

Macros

- #define CSW_REASON_MASK 0xff
- #define **CSW_SCS_FLAG_SHIFT** 16
- #define **CSW_SCS_FLAG_MASK** 0xff0000
- #define **CSW_SCS_FLAG_STRING_MAX** 64
- #define **CSW_REASON_GET_SCS_FLAG**(_reason) (((_reason) & CSW_SCS_FLAG_MASK) >> CSW↩ _SCS_FLAG_SHIFT)
- #define **CSW_REASON_SET_SCS_FLAG**(_scs_flag, _reason) ((((_scs_flag) << CSW_SCS_FLAG_SH↩ IFT) & CSW_SCS_FLAG_MASK) | (_reason))

Enumerations

- enum ieee80211_csw_reason {
  IEEE80211_CSW_REASON_UNKNOWN,
  IEEE80211_CSW_REASON_SCS,
  IEEE80211_CSW_REASON_DFS,
  IEEE80211_CSW_REASON_MANUAL,
  IEEE80211_CSW_REASON_CONFIG,
  IEEE80211_CSW_REASON_SCAN,
  IEEE80211_CSW_REASON_OCAC,
  IEEE80211_CSW_REASON_CSA,
  IEEE80211_CSW_REASON_TDLS_CS,
  IEEE80211_CSW_REASON_MAX }

Functions

- int qcsapi_wifi_start_ocac (const char *ifname, uint16_t channel)

  Start off-channel CAC.
- int qcsapi_wifi_stop_ocac (const char *ifname)

  Stop off-channel CAC.
- int qcsapi_wifi_get_ocac_status (const char *ifname, qcsapi_unsigned_int *status)

  Get the current state of off-channel CAC.
- int qcsapi_wifi_set_ocac_dwell_time (const char *ifname, uint16_t dwell_time)

  Set the dwell time on off-channel for off-channel CAC.
- int qcsapi_wifi_set_ocac_duration (const char *ifname, uint16_t duration)

  Set the duration during which off-channel CAC is running for a DFS channel.
- int qcsapi_wifi_set_ocac_cac_time (const char *ifname, uint16_t cac_time)

  Set the total time on off channel for a DFS channel.
- int qcsapi_wifi_set_ocac_report_only (const char *ifname, uint16_t enable)

  Set the off-channel CAC report only mode.
- int qcsapi_wifi_set_ocac_thrshld (const char *ifname, const char *param_name, uint16_t threshold)

Set the threshold values for OCAC feature.

- int qcsapi_wifi_start_dfs_s_radio (const char *ifname, uint16_t channel)

  Start DFS seamless entry.
- int qcsapi_wifi_stop_dfs_s_radio (const char *ifname)

  Stop DFS seamless entry.
- int qcsapi_wifi_get_dfs_s_radio_status (const char *ifname, qcsapi_unsigned_int *status)

  Get the current status of DFS seamless entry.
- int qcsapi_wifi_get_dfs_s_radio_availability (const char *ifname, qcsapi_unsigned_int *available)

  Get the current availability of DFS seamless entry.
- int qcsapi_wifi_set_dfs_s_radio_dwell_time (const char *ifname, uint16_t dwell_time)

  Set the dwell time on off-channel for DFS seamless entry.
- int qcsapi_wifi_set_dfs_s_radio_duration (const char *ifname, uint16_t duration)

  Set the duration during which DFS seamless entry is running for a DFS channel.
- int qcsapi_wifi_set_dfs_s_radio_wea_duration (const char *ifname, uint32_t duration)

  Set the duration during which DFS seamless entry is running for a weather channel.
- int qcsapi_wifi_set_dfs_s_radio_cac_time (const char *ifname, uint16_t cac_time)

  Set the total time on off channel for a DFS channel.
- int qcsapi_wifi_set_dfs_s_radio_wea_cac_time (const char *ifname, uint32_t cac_time)

  Set the total time on off channel for a weather channel.
- int qcsapi_wifi_set_dfs_s_radio_wea_dwell_time (const char *ifname, uint16_t dwell_time)

  Set the dwell time on off-channel for a weather channel.
- int qcsapi_wifi_set_dfs_s_radio_report_only (const char *ifname, uint16_t enable)

  Set the DFS seamless entry report only mode.
- int qcsapi_wifi_set_dfs_s_radio_thrshld (const char *ifname, const char *param_name, uint16_t threshold)

  Set the threshold values for DFS seamless entry.
- int qcsapi_wifi_set_scs_leavedfs_chan_mtrc_mrgn (const char *ifname, uint8_t leavedfs_chan_mtrc_mrgn)

  Set channel metric margin for SCS channel ranking to switch from DFS-channel to Non-DFS-channel.
- int qcsapi_wifi_get_list_DFS_channels (const char *region_by_name, const int DFS_flag, const qcsapi_unsigned_int bw, string_1024 list_of_channels)

  Get the list of DFS channels.
- int qcsapi_regulatory_get_list_DFS_channels (const char *region_by_name, const int DFS_flag, const qcsapi_unsigned_int bw, string_1024 list_of_channels)

  Get the list of DFS channels.
- int qcsapi_wifi_is_channel_DFS (const char *region_by_name, const qcsapi_unsigned_int the_channel, int *p_channel_is_DFS)

  Is the given channel a DFS channel.
- int qcsapi_regulatory_is_channel_DFS (const char *region_by_name, const qcsapi_unsigned_int the_↩ channel, int *p_channel_is_DFS)

  Is the given channel a DFS channel.
- int qcsapi_wifi_get_dfs_cce_channels (const char *ifname, qcsapi_unsigned_int *p_prev_channel, qcsapi_unsigned_int *p_cur_channel)

  Get previous and current channels from the most recent DFS channel change event.
- int qcsapi_wifi_get_DFS_alt_channel (const char *ifname, qcsapi_unsigned_int *p_dfs_alt_chan)

  Get the alternative DFS channel to be used in case of radar detection.
- int qcsapi_wifi_set_DFS_alt_channel (const char *ifname, const qcsapi_unsigned_int dfs_alt_chan)

  Set the alternative DFS channel to be used in case of radar detection.
- int qcsapi_wifi_start_dfs_reentry (const char *ifname)

  Start a channel scan and select a best DFS channel for usage.
- int qcsapi_wifi_start_scan_ext (const char *ifname, const int scan_flag)

  Start a channel scan and select channel based on given rules.
- int qcsapi_wifi_get_csw_records (const char *ifname, int reset, qcsapi_csw_record *record)

Get channel switch history records.

- int qcsapi_wifi_get_radar_status (const char *ifname, qcsapi_radar_status *rdstatus)

  Get channel radar status and history of detected records.

- int qcsapi_wifi_get_cac_status (const char *ifname, int *cacstatus)

  Get CAC status.

9.20.1 Detailed Description

9.20.2 OCAC

OCAC is a feature where the DFS master device will periodically scan off-channel in order to detect radar on a channel different to the current operating channel. This feature is limited to operation with up to 2 BSSes only.

9.20.3 Macro Definition Documentation

9.20.3.1 CSW_REASON_MASK

```
#define CSW_REASON_MASK 0xff
```

Reason for channel change

9.20.4 Enumeration Type Documentation

9.20.4.1 ieee80211_csw_reason

```
enum ieee80211_csw_reason
```

Enumerator

IEEE80211_CSW_REASON_UNKNOWN	Reason is unknown
IEEE80211_CSW_REASON_SCS	Smart channel selection
IEEE80211_CSW_REASON_DFS	Radar detection
IEEE80211_CSW_REASON_MANUAL	Channel set by user
IEEE80211_CSW_REASON_CONFIG	Configuration change
IEEE80211_CSW_REASON_SCAN	Scan initiated by user
IEEE80211_CSW_REASON_OCAC	Off-channel CAC
IEEE80211_CSW_REASON_CSA	Channel switch announcement
IEEE80211_CSW_REASON_TDLS_CS	TDLS Channel switch announcement
IEEE80211_CSW_REASON_MAX	Number of values

9.20.5 Function Documentation

9.20.5.1 qcsapi_wifi_start_ocac()

```
int qcsapi_wifi_start_ocac (
            const char * ifname,
            uint16_t channel )
```

This API is used to start off-channel CAC on a DFS channel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	specifies the DFS channel for CAC. 0 is to select DFS channel automatically.

Returns

0 on success or negative values on error.

`call_qcsapi` interface:

```
call_qcsapi start_ocac wifi0 {auto | <DFS_channel>}
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_start_dfs_s_radio.

9.20.5.2 qcsapi_wifi_stop_ocac()

```
int qcsapi_wifi_stop_ocac (
            const char * ifname )
```

This API is used to stop off-channel CAC.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi stop_ocac wifi0
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_stop_dfs_s_radio.

9.20.5.3 qcsapi_wifi_get_ocac_status()

```
int qcsapi_wifi_get_ocac_status (
            const char * ifname,
            qcsapi_unsigned_int * status )
```

This API return the current state of off-channel CAC.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
status	value that contains if OCAC is enabled or disabled.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_ocac_status <WiFi interface>
```

The output will be the word "Disabled" or "Enabled" unless an error occurs.

Note

This API is deprecated and replaced with qcsapi_wifi_get_dfs_s_radio_status.

9.20.5.4 qcsapi_wifi_set_ocac_dwell_time()

```
int qcsapi_wifi_set_ocac_dwell_time (
            const char * ifname,
            uint16_t dwell_time )
```

API sets the dwell time for the off-channel CAC feature, ie. the duration on off channel within a beacon interval. Unit is in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dwell_time	Dwell time on off-channel in a beacon interval.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_ocac_dwell_time <WiFi interface> <dwelltime>
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_dfs_s_radio_dwell_time.

9.20.5.5 qcsapi_wifi_set_ocac_duration()

```
int qcsapi_wifi_set_ocac_duration (
            const char * ifname,
            uint16_t duration )
```

API sets the duration during which the off-channel CAC is running for a specified DFS channel Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
duration	Duration for a specified DFS channel to run off-channel CAC.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_ocac_duration <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_dfs_s_radio_duration.

9.20.5.6 qcsapi_wifi_set_ocac_cac_time()

```
int qcsapi_wifi_set_ocac_cac_time (
            const char * ifname,
            uint16_t cac_time )
```

API sets the total time on off channel for a specified DFS channel Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
cac_time	total time on the specified DFS channel.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_ocac_cac_time <WiFi interface> <cac_time>
```

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_dfs_s_radio_cac_time.

9.20.5.7 qcsapi_wifi_set_ocac_report_only()

```
int qcsapi_wifi_set_ocac_report_only (
            const char * ifname,
            uint16_t enable )
```

API sets the off-channel CAC as report only mode, that means, don't switch channel after off-channel CAC is completed if report only mode is set.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable	0 - disable report only mode, otherwise enable it.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

`call_qcsapi set_ocac_report_only <WiFi interface> <1 or 0>`

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_dfs_s_radio_report_only.

9.20.5.8 qcsapi_wifi_set_ocac_thrshld()

```
int qcsapi_wifi_set_ocac_thrshld (
            const char * ifname,
            const char * param_name,
            uint16_t threshold )
```

API sets the threshold for various parameters that control the off-channel CAC feature. Threshold affects the sensitivity of the feature.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param_name	The threshold by name which is to be set
threshold	The value of the threshold to be set

Returns

0 on success or negative values on error

`call_qcsapi` interface:

`call_qcsapi set_ocac_thrshld <WiFi interface> <threshold_name> <value>`
threshold name is one of "fat", "traffic" and "cca_intf". "fat" means the free air time, and the threshold value is the percentage for the free air time. off-channel CAC can run when the current FAT is larger than this threshold. "traffic" is the traffic of local BSS, and the threshold value is the percentage for the local traffic time against the measurement time. off-channel CAC can run when the local traffic is less than the threshold value. "cca_intf" means the cca interference on off channel. AP can switch to the DFS channel after off channel CAC only when no radar is detected on this DFS channel and the cca interference on DFS channel is less than the threshold, which is the percentage for the interference traffic time against the measurement time.

Unless an error occurs, the output will be the string `complete`.

Note

This API is deprecated and replaced with qcsapi_wifi_set_dfs_s_radio_thrshld.

9.20.5.9 qcsapi_wifi_start_dfs_s_radio()

```
int qcsapi_wifi_start_dfs_s_radio (
            const char * ifname,
            uint16_t channel )
```

This API is used to start DFS seamless entry on a DFS channel.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	specifies the DFS channel. 0 is to select DFS channel automatically.

Returns

0 on success or negative values on error.

`call_qcsapi` interface:

```
call_qcsapi start_dfs_s_radio wifi0 {auto | <DFS_channel>}
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.10 qcsapi_wifi_stop_dfs_s_radio()

```
int qcsapi_wifi_stop_dfs_s_radio (
            const char * ifname )
```

This API is used to stop DFS seamless entry.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi stop_dfs_s_radio wifi0
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.11 qcsapi_wifi_get_dfs_s_radio_status()

```
int qcsapi_wifi_get_dfs_s_radio_status (
            const char * ifname,
            qcsapi_unsigned_int * status )
```

This API return the current status of whether DFS seamless entry is started or not.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
status	value that contains if DFS seamless entry is enabled or not.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_dfs_s_radio_status <WiFi interface>
```

The output will be the word "Disabled" or "Enabled" unless an error occurs.

9.20.5.12 qcsapi_wifi_get_dfs_s_radio_availability()

```
int qcsapi_wifi_get_dfs_s_radio_availability (
            const char * ifname,
            qcsapi_unsigned_int * available )
```

This API return the status of whether DFS seamless entry is available or not with the current configuration.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
available	value that contains if DFS seamless entry is available or unavailable.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_dfs_s_radio_availability <WiFi interface>
```

The output will be the word "Available" or "Unavailable" unless an error occurs.

9.20.5.13 qcsapi_wifi_set_dfs_s_radio_dwell_time()

```
int qcsapi_wifi_set_dfs_s_radio_dwell_time (
            const char * ifname,
            uint16_t dwell_time )
```

API sets the dwell time for the DFS seamless entry feature, ie. the duration on off channel within a beacon interval. Unit is in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dwell_time	Dwell time on off-channel in a beacon interval.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_dwell_time <WiFi interface> <dwelltime>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.14 qcsapi_wifi_set_dfs_s_radio_duration()

```
int qcsapi_wifi_set_dfs_s_radio_duration (
            const char * ifname,
            uint16_t duration )
```

API sets the duration during which the DFS seamless entry is running for a specified DFS channel Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
duration	Duration for a specified DFS channel to run DFS seamless entry.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_duration <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.15 qcsapi_wifi_set_dfs_s_radio_wea_duration()

```
int qcsapi_wifi_set_dfs_s_radio_wea_duration (
            const char * ifname,
            uint32_t duration )
```

API sets the duration during which the DFS seamless entry is running for a specified weather channel which is a DFS channel. Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
duration	Duration for a specified DFS channel to run DFS seamless entry.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_wea_duration <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.16 qcsapi_wifi_set_dfs_s_radio_cac_time()

```
int qcsapi_wifi_set_dfs_s_radio_cac_time (
            const char * ifname,
            uint16_t cac_time )
```

API sets the total time on off channel for a specified DFS channel Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
cac_time	total time on the specified DFS channel.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_cac_time <WiFi interface> <cac_time>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.17 qcsapi_wifi_set_dfs_s_radio_wea_cac_time()

```
int qcsapi_wifi_set_dfs_s_radio_wea_cac_time (
            const char * ifname,
            uint32_t cac_time )
```

API sets the total time on off channel for a specified weather channel which is a DFS channel. Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
cac_time	total time on the specified DFS channel.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_wea_cac_time <WiFi interface> <cac_time>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.18 qcsapi_wifi_set_dfs_s_radio_wea_dwell_time()

```
int qcsapi_wifi_set_dfs_s_radio_wea_dwell_time (
            const char * ifname,
            uint16_t dwell_time )
```

API sets the dwell time for a specified weather channel, ie. the duration on off channel within a beacon interval. Unit is in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dwell_time	Dwell time on the specified weather channel in a beacon interval.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_wea_dwell_time <WiFi interface> <dwelltime>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.19 qcsapi_wifi_set_dfs_s_radio_report_only()

```
int qcsapi_wifi_set_dfs_s_radio_report_only (
            const char * ifname,
            uint16_t enable )
```

API sets the DFS seamless entry as report only mode, that means, don't switch channel after DFS seamless entry is completed if report only mode is set.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable	0 - disable report only mode, otherwise enable it.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_dfs_s_radio_report_only <WiFi interface> <1 or 0>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.20 qcsapi_wifi_set_dfs_s_radio_thrshld()

```
int qcsapi_wifi_set_dfs_s_radio_thrshld (
            const char * ifname,
            const char * param_name,
            uint16_t threshold )
```

API sets the threshold for various parameters that control the DFS seamless entry. Threshold affects the sensitivity of the feature.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param_name	The threshold by name which is to be set
threshold	The value of the threshold to be set

Returns

0 on success or negative values on error

`call_qcsapi` interface:

`call_qcsapi set_dfs_s_radio_thrshld <WiFi interface> <threshold_name> <value>`
threshold name is one of "fat", "traffic" and "cca_intf". "fat" means the free air time, and the threshold value is the percentage for the free air time. DFS seamless entry can run when the current FAT is larger than this threshold.

"traffic" is the traffic of local BSS, and the threshold value is the percentage for the local traffic time against the measurement time. DFS seamless entry can run when the local traffic is less than the threshold value. "cca_intf" means the cca interference on off channel. AP can switch to the DFS channel after DFS seamless entry only when no radar is detected on this DFS channel and the cca interference on DFS channel is less than the threshold, which is the percentage for the interference traffic time against the measurement time.

Unless an error occurs, the output will be the string `complete`.

9.20.5.21 qcsapi_wifi_set_scs_leavedfs_chan_mtrc_mrgn()

```
int qcsapi_wifi_set_scs_leavedfs_chan_mtrc_mrgn (
            const char * ifname,
            uint8_t leavedfs_chan_mtrc_mrgn )
```

This API controls the channel metric margin SCS used for channel ranking to switch from DFS-channel to Non-D↩ FS-channel

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
leavedfs_chan_mtrc_mrgn	value that indicates the channel metric margin.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_leavedfs_chan_mtrc_mrgn <WiFi interface> <leavedfs_↩
chan_mtrc_mrgn>
```

Unless an error occurs, the output will be the string `complete`.

9.20.5.22 qcsapi_wifi_get_list_DFS_channels()

```
int qcsapi_wifi_get_list_DFS_channels (
            const char * region_by_name,
            const int DFS_flag,
            const qcsapi_unsigned_int bw,
            string_1024 list_of_channels )
```

Use this API to get a list of all channels that require following the DFS protocols, or alternately a list of channels that do not require the DFS protocols.

Parameters

region_by_name	the region to return. Has the same interpretation as with the regulatory authority APIs.
DFS_flag	set to 1 to get a list of DFS affected channels, set to 0 to get the complement list of channels.
bw	the bandwidth in use - either 20 or 40 to represent 20MHz and 40MHz respectively.
list_of_channels	return parameter to contain the comma delimited list of channels.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_list_DFS_channels <regulatory region> <1 | 0> <20 | 40>
```

where <regulatory region> should be one of the regions listed in by the get list regulatory regions API/command.

Choices for the other two parameters are as shown above.

Unless an error occurs, the output will be a list of channels, each value separated by a comma.

Examples:

To get the list of 40 MHz channels that require following the DFS protocols for Europe, enter:

```
call_qcsapi get_list_DFS_channels eu 1 40
```

To get the list of 20 MHz channels that do not require DFS protocols for the US, enter:

```
call_qcsapi get_list_DFS_channels us 0 20
```

9.20.5.23 qcsapi_regulatory_get_list_DFS_channels()

```
int qcsapi_regulatory_get_list_DFS_channels (
            const char * region_by_name,
            const int DFS_flag,
            const qcsapi_unsigned_int bw,
            string_1024 list_of_channels )
```

Use this API to get a list of all channels that require following the DFS protocols, or alternately a list of channels that do not require the DFS protocols.

Parameters

region_by_name	the region to return. Has the same interpretation as with the regulatory authority APIs.
DFS_flag	set to 1 to get a list of DFS affected channels, set to 0 to get the complement list of channels.
bw	the bandwidth in use - either 20 or 40 to represent 20MHz and 40MHz respectively.
list_of_channels	return parameter to contain the comma delimited list of channels.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_list_DFS_channels <regulatory region> <1 | 0> <20 | 40 |
80>
```

where <regulatory region> should be one of the regions listed in by the get list regulatory regions API/command.

Choices for the other two parameters are as shown above.

Unless an error occurs, the output will be a list of channels, each value separated by a comma.

Examples:

To get the list of 80 MHz channels that require following the DFS protocols for Europe, enter:

```
call_qcsapi get_list_DFS_channels eu 1 80
```

To get the list of 40 MHz channels that require following the DFS protocols for Europe, enter:

```
call_qcsapi get_list_DFS_channels eu 1 40
```

To get the list of 20 MHz channels that do not require DFS protocols for the US, enter:

```
call_qcsapi get_list_DFS_channels us 0 20
```

9.20.5.24 qcsapi_wifi_is_channel_DFS()

```
int qcsapi_wifi_is_channel_DFS (
            const char * region_by_name,
            const qcsapi_unsigned_int the_channel,
            int * p_channel_is_DFS )
```

Use this API to determine whether a particular channel is subject to the DFS protocols.

Parameters

region_by_name	the region to return. Has the same interpretation as with the regulatory authority APIs.
the_channel	unsigned integer from 0 to 255. The channel must be valid for the referenced regulatory region.
p_channel_is_DFS	return value which is set to 1 if the channel is affected by DFS, set to 0 otherwise.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi is_channel_DFS <regulatory region> <channel>
```

where <regulatory region> should be one of the regions listed in by the get list regulatory regions API / command and <channel> is an unsigned integer.

Unless an error occurs, the output will be either 0 or 1 depending on whether DFS protocols are required for the referenced channel.

9.20.5.25 qcsapi_regulatory_is_channel_DFS()

```
int qcsapi_regulatory_is_channel_DFS (
            const char * region_by_name,
            const qcsapi_unsigned_int the_channel,
            int * p_channel_is_DFS )
```

Use this API to determine whether a particular channel is subject to the DFS protocols.

Parameters

region_by_name	the region to return. Has the same interpretation as with the regulatory authority APIs.
the_channel	unsigned integer from 0 to 255. The channel must be valid for the referenced regulatory region.
p_channel_is_DFS	return value which is set to 1 if the channel is affected by DFS, set to 0 otherwise.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi is_channel_DFS <regulatory region> <channel>
```

where <regulatory region> should be one of the regions listed in by the get list regulatory regions API / command and <channel> is an unsigned integer.

Unless an error occurs, the output will be either 0 or 1 depending on whether DFS protocols are required for the referenced channel.

9.20.5.26 qcsapi_wifi_get_dfs_cce_channels()

```
int qcsapi_wifi_get_dfs_cce_channels (
            const char * ifname,
            qcsapi_unsigned_int * p_prev_channel,
            qcsapi_unsigned_int * p_cur_channel )
```

This API returns the channel switched from and to as a result of the most recent DFS channel change event.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_prev_channel	result memory pointer for the channel switched from
p_cur_channel	result memory pointer for the channel switched to

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_dfs_cce_channels <WiFi interface>
```

The output will be the previous channel number then the current channel number, unless an error occurs. If no DFS channel change has occurred, both numbers will be zero.

9.20.5.27 qcsapi_wifi_get_DFS_alt_channel()

```
int qcsapi_wifi_get_DFS_alt_channel (
            const char * ifname,
            qcsapi_unsigned_int * p_dfs_alt_chan )
```

This API call is used to get the alternative DFS channel that will be switched over to in case radar is detected in the current channel. This is known as a 'fast switch', to allow quickly changing to another high power channel without having to do slow scans through all the channels.

Note

This API can only be called on an AP device.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_dfs_alt_chan	return parameter for the alternative DFS channel.

Returns

>= 0 on success, < 0 on error. On success, p_dfs_alt_chan will contain the configured alternate DFS channel.

`call_qcsapi` interface:

```
call_qcsapi get_DFS_alt_channel <WiFi interface>
```

Unless an error occurs, the output will be the string containing the DFS alternative channel (`0` if no alternative channel is specified).

See also

qcsapi_wifi_set_DFS_alt_channel

9.20.5.28 qcsapi_wifi_set_DFS_alt_channel()

```
int qcsapi_wifi_set_DFS_alt_channel (
            const char * ifname,
            const qcsapi_unsigned_int dfs_alt_chan )
```

This API call is used to set the alternative DFS channel that will be switched over to in case radar is detected in the current channel. This is known as a 'fast switch', to allow quickly changing to another high power channel without having to do slow scans through all the channels.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dfs_alt_chan	the alternative DFS channel to set.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_DFS_alt_channel <WiFi interface> <alternative channel>
```

Unless an error occurs, the output will be the string `complete`.

Error can occur if the alternative channel being set is the same as the current channel on the device.

See also

qcsapi_wifi_get_DFS_alt_channel

9.20.5.29 qcsapi_wifi_start_dfs_reentry()

```
int qcsapi_wifi_start_dfs_reentry (
            const char * ifname )
```

This API is used to trigger a channel scan and once the scan is done, the AP will switch to a DFS channel based on channel ranking algorithm.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

`call_qcsapi` interface:

`call_qcsapi start_dfsreentry <WiFi interface>` Unless an error occurs, the output will be the string `complete`.

Error can occur if all DFS channels have been in non-occupy list.

9.20.5.30 qcsapi_wifi_start_scan_ext()

```
int qcsapi_wifi_start_scan_ext (
            const char * ifname,
            const int scan_flag )
```

This API provides a way to scan channel and select channel by different rules.
The value of `scan_flag` specifies the parameters for this API, including the control flags for scanning activity, the channel set which is used for channel selection, and the algorithm which is used to pick the channel.

Flags used as channel set are defined as below and they are mutually-exclusive:

```
IEEE80211_PICK_ALL          0x0001
IEEE80211_PICK_DFS          0x0002
IEEE80211_PICK_NONDFS               0x0004
```

Flags used as the control flags for scanning activity as below, the flags which have string "BG" are mutually-exclusive:

```
IEEE80211_PICK_SCAN_FLUSH           0x0008
IEEE80211_PICK_BG_ACTIVE            0x0010
IEEE80211_PICK_BG_PASSIVE_FAST              0x0020
IEEE80211_PICK_BG_PASSIVE_NORMAL    0x0040
IEEE80211_PICK_BG_PASSIVE_SLOW              0x0080
```

Flags used as algorithm are defined as below and they are mutually-exclusive:

```
IEEE80211_PICK_CLEAREST             0x0100
IEEE80211_PICK_REENTRY              0x0200
IEEE80211_PICK_NOPICK               0x0400
IEEE80211_PICK_NOPICK_BG    0x0800
```

`scan_flag` may be any combination of channel set and algorithm. The header file `net80211/ieee80211_dfs_reentry.h` including this macros comes with the package libqcsapi_client_src.zip.

`call_qcsapi` interface:

`call_qcsapi start_scan wifi0 <algorithm> <select_channel> <control_flag>`
Where `<algorithm>` should be "reentry", "clearest", "no_pick" or "background". "reentry" means it will start dfs-reentry function. "clearest" means it will pick the clearest channel. "no_pick" means it will only perform channel scan. "background" means scan channel in the background and no_pick.

Where `<select_channel>` should be "dfs", "non_dfs" or "all". This parameter indicates that what kind of channel to be selected. With using "dfs", It will pick channel from available dfs channels. With using "non-dfs", it will pick channel from available non-dfs channels. And "all" is default which means it will pick channel from all available channels.

Where `<control_flags>` should be "flush", and/or "active", "fast", "normal" and "slow". Theses parameter indicates the required behaviors for scanning activity. If "flush" is set, the previous scanning result will be flushed at first before the new channel scanning. "active", "fast", "normal" and "slow" work for "background" algorithm only, and only one of them can be set. "active" mean the active scanning on DFS channel, and others mean the passive scanning on DFS channels. "fast" means the fast passive scanning, "slow" means the slow passive scanning, and "normal" is between of them.

`call_qcsapi` interface:

```
call_qcsapi start_scan wifi0 reentry dfs
```

9.20.5.31 qcsapi_wifi_get_csw_records()

```
int qcsapi_wifi_get_csw_records (
            const char * ifname,
            int reset,
            qcsapi_csw_record * record )
```

This API reports back the channel change history up to a maximum of 32 records. This API can also reset the records. As a get function, it needs `struct qcsapi_csw_record` as a buffer to receive return data.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
reset	indicate whether to reset the records. "1" to reset records, and "0" to get records.
record	where to store the records.

Returns

>=0 on success, <0 on error.

Note

This API does not work on a STA.

`call_qcsapi` interface:

```
call_qcsapi get_csw_records <WiFi interface>
call_qcsapi get_csw_records <WiFi interface> 1
```

The output from the first command is the channel change record count, followed by a list of channel change records. A channel change record includes time from start-up, channel that was selected and a reason for channel change. Reasons are enumerated by ieee80211_csw_reason. Mappings to printed strings are defined by the array `qcsapi_csw_reason_list`.

The output from the second command is the channel change history followed by the string "`clear records complete`".

```
#call_qcsapi get_csw_records wifi0 1
channel switch history record count : 3
time=1234 channel=123 reason=SCS
time=11 channel=36 reason=CONFIG
time=7 channel=40 reason=CONFIG
clear records complete
```

9.20.5.32 qcsapi_wifi_get_radar_status()

```
int qcsapi_wifi_get_radar_status (
            const char * ifname,
            qcsapi_radar_status * rdstatus )
```

This API is used to query the status of a DFS channel; whether it is in non-occupy list, and how many times the radar signal has be detected on this channel. This data can be used to analyse the local enviroment for radar usage.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
rdstatus	when used as input,it contain channel to query, when used as return value, it stores the radar channel status.

Returns

>=0 on success, <0 on error.

Note

The channel passed in the rdstatus structure must be DFS affected, based on the regulatory region.

`call_qcsapi` interface:

```
call_qcsapi get_radar_status <WiFi interface> <DFS-Channel>
```

Unless an error occurs, the output will show the radar status of `DFS-Channel`.
Example:

```
#call_qcsapi get_radar_status wifi0 100
channel 100:
radar_status=0
radar_count=1
```

9.20.5.33 qcsapi_wifi_get_cac_status()

```
int qcsapi_wifi_get_cac_status (
            const char * ifname,
            int * cacstatus )
```

This API is used to get CAC status on AP. Application can use this API to poll CAC status and ensure CAC process is completed.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
cacstatus	return the currently CAC status, 1 for CAC is running and 0 for no CAC is running.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_cacstatus <WiFi interface>
```

Unless an error occurs, the output will be the 1 or 0.

9.21 Scan APIs

This section describes APIs that report on properties of APs that were found when the STA scanned the WiFi channels.
When a WiFi device configured as a STA starts up, it typically scans the WiFi channels for APs in its neighborhood.

Functions

- int qcsapi_wifi_get_results_AP_scan (const char *ifname, qcsapi_unsigned_int *p_count_APs)

  Get the results of an AP scan.
- int qcsapi_wifi_get_count_APs_scanned (const char *ifname, qcsapi_unsigned_int *p_count_APs)

  Get a count of the number of APs scanned.
- int qcsapi_wifi_get_properties_AP (const char *ifname, const qcsapi_unsigned_int index_AP, qcsapi_ap_properties *p_ap_properties)

  Get AP properties per scan result.
- int qcsapi_wifi_set_scan_chk_inv (const char *ifname, int scan_chk_inv)

  Set scan results check interval, unit is second.
- int qcsapi_wifi_get_scan_chk_inv (const char *ifname, int *p)

  Get scan results check interval, unit is second.
- int qcsapi_wifi_set_scan_buf_max_size (const char *ifname, const unsigned int max_buf_size)

  Set the maximum scan buffer size for returned scan results.
- int qcsapi_wifi_get_scan_buf_max_size (const char *ifname, unsigned int *max_buf_size)

  Get the maximum scan buffer size for returned scan results.
- int qcsapi_wifi_set_scan_table_max_len (const char *ifname, const unsigned int max_table_len)

  Set the maximum number of returned scan results.
- int qcsapi_wifi_get_scan_table_max_len (const char *ifname, unsigned int *max_table_len)

  Get the maximum number of returned scan results.
- int qcsapi_wifi_set_dwell_times (const char *ifname, const unsigned int max_dwell_time_active_chan, const unsigned int min_dwell_time_active_chan, const unsigned int max_dwell_time_passive_chan, const unsigned int min_dwell_time_passive_chan)

  Set dwell times.
- int qcsapi_wifi_get_dwell_times (const char *ifname, unsigned int *p_max_dwell_time_active_chan, unsigned int *p_min_dwell_time_active_chan, unsigned int *p_max_dwell_time_passive_chan, unsigned int *p_min_dwell_time_passive_chan)

  Get dwell times.
- int qcsapi_wifi_set_bgscan_dwell_times (const char *ifname, const unsigned int dwell_time_active_chan, const unsigned int dwell_time_passive_chan)

  Set bgscan dwell times.
- int qcsapi_wifi_get_bgscan_dwell_times (const char *ifname, unsigned int *p_dwell_time_active_chan, unsigned int *p_dwell_time_passive_chan)

  Get bgscan dwell times.
- int qcsapi_wifi_start_scan (const char *ifname)

  Start a scan on the given wireless interface.
- int qcsapi_wifi_cancel_scan (const char *ifname, int force)

  Cancel an ongoing scan on the given wireless interface.
- int qcsapi_wifi_get_scan_status (const char *ifname, int *scanstatus)

  Get scan status.
- int qcsapi_wifi_enable_bgscan (const char *ifname, const int enable)

  Enable background scan.
- int qcsapi_wifi_get_bgscan_status (const char *ifname, int *enable)

  get background scan status

- int qcsapi_wifi_wait_scan_completes (const char *ifname, time_t timeout)

  Wait until the currently running scan has completed.
- int qcsapi_wifi_set_threshold_of_neighborhood_type (const char *ifname, uint32_t type, uint32_t threshold)

  Set the threshold of neighborhood density type.
- int qcsapi_wifi_get_threshold_of_neighborhood_type (const char *ifname, uint32_t type, uint32_t *threshold)

  Get the threshold of neighborhood density type.
- int qcsapi_wifi_get_neighborhood_type (const char *ifname, uint32_t *type, uint32_t *count)

  Get the type of neighborhood density.

9.21.1 Detailed Description

Note

The list of channels will be limited to those legal in the local regulatory region by calling the Set Regulatory Region API.

Two APIs are available to report on the results of the last AP scan. The first API gets the results and caches them in memory. A second API then reports the properties of a particular AP. The AP is selected by index, starting at 0. An application that wants to examine all APs can call Get Properties AP (see qcsapi_wifi_get_properties_AP), starting first with the index set to 0, and incrementing that index in each subsequent call until the API returns an error (-ERANGE).

If either API is called on a WiFI device configured as an AP, the API will return an error (see enum qcsapi_only_↩ on_STA).

Note

The Get Properties API uses the results it finds in the in-memory cache. To insure the results from the latest AP scan are used, an application should always call the get results AP scan API first (see qcsapi_wifi_get_↩ results_AP_scan). The example application shows how this should be programmed.

9.21.2 Data Structure to Report the Properties of an AP

The properties of an AP are returned in the data structure shown below (see struct qcsapi_ap_properties for full details):

```
typedef struct qcsapi_ap_properties
{
   qcsapi_SSID             ap_name_SSID;
   qcsapi_mac_addr         ap_mac_addr;
   qcsapi_unsigned_int     ap_flags;
   int                     ap_channel;
   int                     ap_RSSI;
   int                     ap_protocol;
   int                     ap_encryption_modes;
   int                     ap_authentication_mode;
   int                     ap_best_rate;
   int                     ap_wps;
   int                     ap_80211_proto;
} qcsapi_ap_properties;
```

As can be seen from the data struct definition, properties returned by the Get AP Properties API include its SSID, its MAC address, what channel it is broadcasting beacons on and the relative signal strength (RSSI) from that AP. RSSI will range from 1 up to 70; the larger this value, the stronger the signal from the AP is on the local STA.

Flags (ap_flags) is a bit mask and currently only reports on whether the AP has enabled security. If the low-order bit (0x01) is set, the referenced AP has enabled security; if that bit is cleared, the referenced AP has disabled security. Remaining bits have no meaning currently.

If security is reported as enabled, the security protocol(s) in use, the encryption mode(s) and the authentication mode are all reported.
Possible values for the protocol(s) (ap_protocol) are derived from these two values:

```
qcsapi_protocol_WPA_mask = 1
qcsapi_protocol_11i_mask = 2
```

Since an AP can enable both WPA and 11i (WPA2) at the same time, the two values are actually bit masks.
Possible values for the authentication mode (ap_encryption_modes) are shown below:

```
qcsapi_ap_PSK_authentication = 1
qcsapi_ap_EAP_authentication = 2
```

Possible values for the encryption mode(s) (ap_authentication_mode) are derived from these two values:

```
qcsapi_ap_TKIP_encryption_mask = 0x01
qcsapi_ap_CCMP_encryption_mask = 0x02
```

Since an AP can enable both CCMP and TKIP as encryption modes at the same time, the two values are actually bit masks.
The value of WPS capability (ap_wps) is 1 when AP supports WPS, or 0 when not.
The value of IEEE802.11 protocol (ap_80211_proto) may be any combination of following values:

```
802.11b    0x01
802.11g    0x02
802.11a    0x04
802.11n    0x08
```

9.21.3 Demonstration Application for AP Scan APIs

A demonstration command line application, `show_access_points`, is included with the SDK.

This application details how to use the Get Results AP Scan API before getting the properties for individual APs.

Look at the code fragment below:

```
errorval = qcsapi_wifi_get_results_AP_scan( ifname, &count_APs );
if (errorval >= 0) {
   qcsapi_unsigned_int     iter;
   qcsapi_ap_properties    ap_properties;
   for (iter = 0; iter < count_APs && errorval >= 0; iter++) {
           errorval = qcsapi_wifi_get_properties_AP( ifname, iter, &ap_properties );
           if (errorval >= 0) {
                   show_ap_properties( iter + 1, &ap_properties );
           }
   }
} else {
   /* come here if the initial call to get the results of the AP scan fails */
}
```

Notice the program first calls qcsapi_wifi_get_results_AP_scan before looping over the individual APs, calling qcsapi_wifi_get_properties_AP to get the properties for individual APs. Such a programming model is recommended whenever an application reviews or lists the properties of individual APs.

9.21.4 Function Documentation

9.21.4.1 qcsapi_wifi_get_results_AP_scan()

```
int qcsapi_wifi_get_results_AP_scan (
            const char * ifname,
            qcsapi_unsigned_int * p_count_APs )
```

This API gets the results of the most recent AP scan and caches them in memory for future reference.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_count_APs	return parameter to contain the count of the number of AP scan results.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_results_AP_scan <WiFi interface>
```

Unless an error occurs, the output will be the number of APs found in the last scan.

9.21.4.2 qcsapi_wifi_get_count_APs_scanned()

```
int qcsapi_wifi_get_count_APs_scanned (
            const char * ifname,
            qcsapi_unsigned_int * p_count_APs )
```

This API call is used to get the count of APs that have been scanned in the most recent channel scan.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_count_APs	return parameter to contain the count of APs that have been scanned in the most recent scan.

Returns

>= 0 on success, < 0 on error. If success, p_count_APs contains a count of the number of APs scanned in the previous scan.

`call_qcsapi` interface:

```
call_qcsapi get_count_APs_scanned <WiFi interface>
```

Unless an error occurs, the output will be the number of APs that were scanned in the previous channel scan.

9.21.4.3 qcsapi_wifi_get_properties_AP()

```
int qcsapi_wifi_get_properties_AP (
            const char * ifname,
            const qcsapi_unsigned_int index_AP,
            qcsapi_ap_properties * p_ap_properties )
```

This API reports on the properties of an AP, with the AP indentified by index. The index is numbered starting at 0.

If the cache of AP scan results is not present, this API will call the Get Results AP Scan to update the cache.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
index_AP	the index to get the result from.
p_ap_properties	return parameter for storing the AP scan properties.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_properties_AP <WiFi interface> <index>
```

Unless an error occurs, the output will be the properties of the referenced AP, in the order of SSID, MAC address, WiFi channel, RSSI, flags, protocol, authentication mode, encryption mode, Qhop flags, WPS flags and maximum supported bandwidth. The SSID is enclosed in quotes since it can have embedded blanks.

WPS flags: bit 0 - WPS supported bit 1 - WPS registrar bit 2 - WPS registrar supporting push-button bit 3 - WPS push-button active

Example output from call_qcsapi:

```
quantenna # call_qcsapi get_properties_AP wifi0 0
"FAE-Lab-10" c0:3f:0e:7d:5a:dc 40 25 0 0 0 0 0 1 40
```

This AP has "FAE-Lab-10" as its SSID (the SSID is enclosed in quotes since an SSID can have embedded blank characters), MAC address of c0:3f:03:7d:51:dc, is broadcasting on WiFi channel 40 (5.2 GHz), with an RSSI of 25. Security is disabled for the AP. Qhop is off and WPS is supported, maximum supported bandwidth is 40M.

```
quantenna # call_qcsapi get_properties_AP wifi0 2
"Quantenna1" 00:26:86:00:11:5f 60 56 1 2 1 2 0 15 80
```

This AP has "Quantenna1" as its SSID, MAC address of 00:26:86:00:11:5f, is broadcasting on WiFi channel 60 (5.3 GHz), with an RSSI of 56. Security is enabled for this AP. The security protocol is WPA2 (11i); the authentication mode is PSK; and the encryption mode is CCMP. Qhop is off, WPS is available and WPS push-button is currently active, maximum supported bandwidth is 80M.

```
quantenna # call_qcsapi get_properties_AP wifi0 4
QCS API error 34: Parameter value out of range
```

When the index becomes too large, the call_qcsapi command will fail as shown above. In this setup, only 4 APs were found in the scan. Since the index is numbered starting at 0, valid index values here are 0, 1, 2 and 3.

Note

The Get Properties API uses the results it finds in the in-memory cache. To ensure the results from the latest AP scan are used, an application should always call the Get Results AP Scan API first. The example application shows how this should be programmed.

See also

qcsapi_ap_properties

9.21.4.4 qcsapi_wifi_set_scan_chk_inv()

```
int qcsapi_wifi_set_scan_chk_inv (
            const char * ifname,
            int scan_chk_inv )
```

This API sets the scan results check interval

Note

primarywifi

Parameters

ifname	the primary WiFi interface, wifi0 only.
scan_chk_inv	interval for scan results availability check

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi set_scan_chk_inv <WiFi interface> <scan_chk_inv>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.5 qcsapi_wifi_get_scan_chk_inv()

```
int qcsapi_wifi_get_scan_chk_inv (
            const char * ifname,
            int * p )
```

This API gets the scan results check interval

Note

This API can only be used on the primary interface (wifi0)

This API is available on AP/STA mode.

Parameters

ifname	the primary WiFi interface, wifi0 only.
p	pointer to interval for scan results check

Returns

A negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_scan_chk_inv <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.6 qcsapi_wifi_set_scan_buf_max_size()

```
int qcsapi_wifi_set_scan_buf_max_size (
            const char * ifname,
            const unsigned int max_buf_size )
```

Configure the maximum buffer size for scan results. If the result list exceeds this size it is sorted according to the following rules prior to truncation.

- matched SSID
- WPS active
- WPA/RSN security
- High RSSI

This API can only be used in STA mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_buf_size	max buffer size vlaue

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_scan_buf_max_size <wifi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.7 qcsapi_wifi_get_scan_buf_max_size()

```
int qcsapi_wifi_get_scan_buf_max_size (
            const char * ifname,
            unsigned int * max_buf_size )
```

This API call is used to retrieve the maximum scan buffer size

This API can only be used in STA mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_buf_size	return value to store scan buffer max size

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_scan_buf_max_size <wifi interface>
```

Unless an error occurs, the output will be the max scan buffer size.

9.21.4.8 qcsapi_wifi_set_scan_table_max_len()

```
int qcsapi_wifi_set_scan_table_max_len (
            const char * ifname,
            const unsigned int max_table_len )
```

This API call is used to set the maximum number of returned scan results If the result list exceeds this number it is sorted according to the following rules prior to truncation.

- matched SSID
- WPS active
- WPA/RSN security
- High RSSI

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_table_len	scan table max length

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_scan_table_max_len <wifi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.9 qcsapi_wifi_get_scan_table_max_len()

```
int qcsapi_wifi_get_scan_table_max_len (
            const char * ifname,
            unsigned int * max_table_len )
```

This API call is used to get the maximum number of returned scan results

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_table_len	return value to store scan table max length

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_scan_table_max_len <wifi interface>
```

Unless an error occurs, the output will be the scan table max length

9.21.4.10 qcsapi_wifi_set_dwell_times()

```
int qcsapi_wifi_set_dwell_times (
            const char * ifname,
            const unsigned int max_dwell_time_active_chan,
            const unsigned int min_dwell_time_active_chan,
            const unsigned int max_dwell_time_passive_chan,
            const unsigned int min_dwell_time_passive_chan )
```

This API sets minimum and maximum active and passive channel dwell times used when scanning.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_dwell_time_active_chan	Maximum dwell time for active scans
min_dwell_time_active_chan	Minimum dwell time for active scans
max_dwell_time_passive_chan	Maximum dwell time for passive scans
min_dwell_time_passive_chan	Maximum dwell time for passive scans

All units are milliseconds.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

call_qcsapi interface:

```
call_qcsapi set_dwell_times <WiFi interface> max_active min_active max_↩
passive min_passive
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.11 qcsapi_wifi_get_dwell_times()

```
int qcsapi_wifi_get_dwell_times (
            const char * ifname,
            unsigned int * p_max_dwell_time_active_chan,
            unsigned int * p_min_dwell_time_active_chan,
            unsigned int * p_max_dwell_time_passive_chan,
            unsigned int * p_min_dwell_time_passive_chan )
```

This API retrieves dwell times from the WLAN driver.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_max_dwell_time_active_chan	Result memory for maximum dwell time for active scans
p_min_dwell_time_active_chan	Result memory for minimum dwell time for active scans
p_max_dwell_time_passive_chan	Result memory for maximum dwell time for passive scans
p_min_dwell_time_passive_chan	Result memory for maximum dwell time for passive scans

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

call_qcsapi interface:

```
call_qcsapi get_dwell_times <WiFi interface>
```

call_qcsapi will print dwell times in argument order to stdout on success, or print an error message to stdout on failure.

9.21.4.12 qcsapi_wifi_set_bgscan_dwell_times()

```
int qcsapi_wifi_set_bgscan_dwell_times (
            const char * ifname,
            const unsigned int dwell_time_active_chan,
            const unsigned int dwell_time_passive_chan )
```

This API sets active and passive channel dwell times used when background scanning.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
dwell_time_active_chan	dwell time for active scans
dwell_time_passive_chan	dwell time for passive scans

All units are milliseconds.

Note

bgscan dwell times should be less than regular scan dwell times.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_bgscan_dwell_times <WiFi interface> active passive
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.13 qcsapi_wifi_get_bgscan_dwell_times()

```
int qcsapi_wifi_get_bgscan_dwell_times (
            const char * ifname,
            unsigned int * p_dwell_time_active_chan,
            unsigned int * p_dwell_time_passive_chan )
```

This API retrieves background scan dwell times from the WLAN driver.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_dwell_time_active_chan	Result memory for dwell time for active scans
p_dwell_time_passive_chan	Result memory for dwell time for passive scans

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_bgscan_dwell_times <WiFi interface>
```

call_qcsapi will print dwell times in argument order to stdout on success, or print an error message to stdout on failure.

9.21.4.14 qcsapi_wifi_start_scan()

```
int qcsapi_wifi_start_scan (
            const char * ifname )
```

This API causes the STA to scan available WiFi channels for beacons and associate with an Access Point whose SSID is configured correctly.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.

Returns

0 if the command succeeded (the scan is triggered, not that the scan is complete).

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi start_scan <WiFi interface>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.15 qcsapi_wifi_cancel_scan()

```
int qcsapi_wifi_cancel_scan (
            const char * ifname,
            int force )
```

This API will cancel any ongoing WiFi channels scanning performed by the STA. It will do nothing if no scanning is currently running.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
force	Set to 0 to trigger scanning cancellation as soon as possible and return immediately. Set to 1 to cancel scan immediately and then return.

Returns

0 if the command succeeded - scan cancellation is triggered (force=0), or scan is cancelled (force=1), or no scan was in progress.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi cancel_scan <WiFi interface> <force>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.16 qcsapi_wifi_get_scan_status()

```
int qcsapi_wifi_get_scan_status (
            const char * ifname,
            int * scanstatus )
```

This API is used to get scan status. Application can use this API to poll scan status and ensure scan is completed.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
scanstatus	return the currently scan status, 1 for scan is running and 0 for no scan is running.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_scanstatus <WiFi interface>
```

Unless an error occurs, the output will be the 1 or 0.

9.21.4.17 qcsapi_wifi_enable_bgscan()

```
int qcsapi_wifi_enable_bgscan (
            const char * ifname,
            const int enable )
```

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
enable	Enable parameter, 1 means enable else 0 means disable

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enable_bgscan <WiFi interface> <enable>
```

where `WiFi interface` is the primary interface, `enable` is 0 | 1.

Unless an error occurs, the output will be the string `complete`.

9.21.4.18 qcsapi_wifi_get_bgscan_status()

```
int qcsapi_wifi_get_bgscan_status (
            const char * ifname,
            int * enable )
```

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
enable	background scan enable status

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_bgscan_status <WiFi interface>
```

where `WiFi interface` is the primary interface, Unless an error occurs, the output will be the Extender related parameter value.

9.21.4.19 qcsapi_wifi_wait_scan_completes()

```
int qcsapi_wifi_wait_scan_completes (
            const char * ifname,
            time_t timeout )
```

This API, when called, will block the calling thread until the previously triggered scan completes.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
timeout	how long to wait for the scan to complete.

If the scan has not completed in the specified timeout interval, the API will return an error reporting timeout.

This API is targeted for the STA but will also work on an AP.

If no scan is in progress, the API will block the calling process until the timeout expires.

Note

To check whether scan is completed in RPC case, use non-block polling API `qcsapi_wifi_get_scan_status()` instead of this block API.

`call_qcsapi` interface:

```
call_qcsapi wait_scan_completes <WiFi interface> <timeout>
```

Unless an error occurs, the output will be the string `complete`.

A timeout will be reported as `QCSAPI error 62:  Timer expired.`

9.21.4.20 qcsapi_wifi_set_threshold_of_neighborhood_type()

```
int qcsapi_wifi_set_threshold_of_neighborhood_type (
            const char * ifname,
            uint32_t type,
            uint32_t threshold )
```

This API call is used to set the threshold of neighborhood density type.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	density type, there are three types, IEEE80211_NEIGHBORHOOD_TYPE_SPARSE = 0, IEEE80211_NEIGHBORHOOD_TYPE_DENSE = 1, IEEE80211_NEIGHBORHOOD_TYPE_VERY_DENSE = 2.
threshold	when AP count belows or equals to this value, driver evaluates the neighborhood as the type above.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_threshold_of_neighborhood_type <WiFi interface> <type>
<threshold>
```

Unless an error occurs, the output will be the string `complete`.

9.21.4.21 qcsapi_wifi_get_threshold_of_neighborhood_type()

```
int qcsapi_wifi_get_threshold_of_neighborhood_type (
            const char * ifname,
            uint32_t type,
            uint32_t * threshold )
```

This API call is used to get the threshold of neighborhood density type.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	density type, there are three types, IEEE80211_NEIGHBORHOOD_TYPE_SPARSE = 0, IEEE80211_NEIGHBORHOOD_TYPE_DENSE = 1, IEEE80211_NEIGHBORHOOD_TYPE_VERY_DENSE = 2.
threshold	a pointer to hold the value of the threshold

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_threshold_of_neighborhood_type <WiFi interface> <type>
```

Unless an error occurs, the output will be the value of the threshold.

9.21.4.22 qcsapi_wifi_get_neighborhood_type()

```
int qcsapi_wifi_get_neighborhood_type (
            const char * ifname,
            uint32_t * type,
            uint32_t * count )
```

This API call is used to get the type of neighborhood density.

Note

This API can only be used in AP mode.

make sure that a scan has been scheduled lately to get a updated result.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	result of type, there are three types, IEEE80211_NEIGHBORHOOD_TYPE_SPARSE = 0, IEEE80211_NEIGHBORHOOD_TYPE_DENSE = 1, IEEE80211_NEIGHBORHOOD_TYPE_VERY_DENSE = 2.
count	neighbor count.

Returns

0 if the configuration file path was updated successfully.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_neighborhood_type <WiFi interface>
```

Unless an error occurs, the output will be one of `"Sparse ([n] neighbor APs)"`, `"Dense ([n] neighbor APs)"`, `"Very dense ([n] neighbor APs)"` or `"Unknown, may need a new scan"`.

9.22 Security Mismatch Backoff APIs

Two APIs are available on the WiFi station (WiFi mode is "Station") to manage retries when a mismatch in security is discovered with its partner Access Point. Because the STA can eventually back off and stop attempting to associate for a period of time, these APIs are referred to as Backoff APIs.
These API configure the number of time to try before backing off and the amount of time to wait before trying again after backing off.

Functions

- int qcsapi_wifi_backoff_fail_max (const char *ifname, const int fail_max)

  Configure the retry backoff failure maximum count.
- int qcsapi_wifi_backoff_timeout (const char *ifname, const int timeout)

  Configure retry backoff timeout.

9.22.1 Detailed Description

9.22.2 Function Documentation

9.22.2.1 qcsapi_wifi_backoff_fail_max()

```
int qcsapi_wifi_backoff_fail_max (
            const char * ifname,
            const int fail_max )
```

Sets the number of times an association attempt can fail before backing off.

Note

This API is only valid on an STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
fail_max	the maximum number of failures permitted. The parameter can range from 2 to 20. The default failure value is 3.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi backoff_fail_max <WiFi interface> <max failure count>
```

Example:

```
call_qcsapi backoff_fail_max wifi0 3
```

Unless an error occurs, the output will be the string `complete`.

9.22.2.2 qcsapi_wifi_backoff_timeout()

```
int qcsapi_wifi_backoff_timeout (
            const char * ifname,
            const int timeout )
```

Configures the time to wait in seconds after backing off before attempting to associate again.

Note

This API is only valid on an STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
timeout	the timeout between backoff and attempting a reconnection. Range is between 10 and 300 seconds. Default value is 60 seconds.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi backoff_timeout <WiFi interface> <timeout>
```

Example:

```
call_qcsapi backoff_timeout wifi0 60
```

Unless an error occurs, the output will be the string `complete`.

9.23 Engineering and Test APIs

These APIs are not expected to be called in normal programming. They are present to assist with engineering testing and performance evaluation.

Functions

- int qcsapi_wifi_get_mcs_rate (const char *ifname, qcsapi_mcs_rate current_mcs_rate)

  Get the current MCS rate.
- int qcsapi_wifi_set_mcs_rate (const char *ifname, const qcsapi_mcs_rate new_mcs_rate)

  Set the MCS rate.
- int qcsapi_wifi_set_pairing_id (const char *ifname, const char *pairing_id)

  Set pairing ID for pairing protection.
- int qcsapi_wifi_get_pairing_id (const char *ifname, char *pairing_id)

  Get pairing ID for pairing protection.
- int qcsapi_wifi_set_pairing_enable (const char *ifname, const char *enable)

  Set pairing enable flag for pairing protection.
- int qcsapi_wifi_get_pairing_enable (const char *ifname, char *enable)

  Get pairing enable flag for pairing protection.
- int qcsapi_non_wps_set_pp_enable (const char *ifname, uint32_t ctrl_state)

  Set non_WPS pairing ptrotection enable flag for pairing protection.
- int qcsapi_non_wps_get_pp_enable (const char *ifname, uint32_t *ctrl_state)

  Get non_WPS pairing ptrotection enable flag for pairing protection.
- int qcsapi_wifi_set_vendor_fix (const char *ifname, int fix_param, int value)

  Set various fix items for compatibility issue with other vendor chipset.
- int qcsapi_errno_get_message (const int qcsapi_retval, char *error_msg, unsigned int msglen)

  Convert a numeric error code to a descriptive string.
- int qcsapi_wifi_get_vco_lock_detect_mode (const char *ifname, unsigned int *p_jedecid)

  get vco lock detect status start/stop.
- int qcsapi_wifi_set_vco_lock_detect_mode (const char *ifname, unsigned int *p_jedecid)

  set vco lock detect start/stop.
- int qcsapi_get_core_dump_size (uint32_t *core_dump_size)

  This API is used to get the size of the core dump that is stored across reboot.
- int qcsapi_get_core_dump (string_4096 buf, uint32_t bytes_to_copy, uint32_t start_offset, uint32_t *bytes_copied)

  This API is used to get the core dump across reboot.
- int qcsapi_get_app_core_dump_size (char *file, uint32_t *core_dump_size)

  This API is used to get the size of the application's core dump file.
- int qcsapi_get_app_core_dump (char *file, string_4096 buf, uint32_t bytes_to_copy, uint32_t offset, uint32_t *bytes_copied)

  This API is used to get the application's core dump file.
- int qcsapi_set_log_level (const char *ifname, qcsapi_log_module_name index, const string_128 params)

  Set the log levels depends on module and level.
- int qcsapi_get_log_level (const char *ifname, qcsapi_log_module_name index, string_128 params)

  Get the current log level for the given module.
- int qcsapi_set_remote_logging (qcsapi_remote_log_action action_type, qcsapi_unsigned_int ipaddr)

  Enable/Disable remote logging to the NPU.
- int qcsapi_set_console (qcsapi_console_action action_type)

  Enable/Disable console.
- int qcsapi_do_system_action (const string_32 action)

Perform the specified system action.

- int qcsapi_wifi_set_max_boot_cac_duration (const char *ifname, const int max_boot_cac_duration)

  Set the maximum boot time CAC duration in seconds.

- int qcsapi_wifi_set_br_isolate (qcsapi_br_isolate_cmd cmd, uint32_t arg)

  Configure bridge isolation.

- int qcsapi_wifi_get_br_isolate (uint32_t *result)

  get br_isolate status

9.23.1 Detailed Description

9.23.2 Function Documentation

9.23.2.1 qcsapi_wifi_get_mcs_rate()

```
int qcsapi_wifi_get_mcs_rate (
            const char * ifname,
            qcsapi_mcs_rate current_mcs_rate )
```

Get the current MCS rate.

Value will be a string with format "MCSn" or "MCSnn", where n or nn is an integer in ASCII format from 0 to 76, excluding 32. For 11ac rate, the value will be string with format "MCSx0y", where x is from 1 to 4, and y is from 0 to 9. x means Nss (number of spatial stream) and y mean MCS index.

If the autorate fallback option has been selected, this API will return Configuration Error.

This API only returns an actual MCS rate if the set MCS rate API has been called to select a particular MCS rate.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
current_mcs_rate	return parameter for storing the current MCS rate.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_mcs_rate <WiFi interface>
```

Unless an error occurs, the output will be an MCS index string, e.g. for 11n, `MCS0, MCS8, MCS76`, for 11ac, `MCS100, MCS307` etc, or `Configuration Error` if the auto rate fallback option has been selected.

This command can return incorrect results if the rate has never been configured.

9.23.2.2 qcsapi_wifi_set_mcs_rate()

```
int qcsapi_wifi_set_mcs_rate (
            const char * ifname,
            const qcsapi_mcs_rate new_mcs_rate )
```

Set the current MCS rate. For 11n rate, value is required to be a string with format "MCSn" or "MCSnn", where n or nn is an integer in ASCII format from 0 to 76, excluding 32. Leading zeros are NOT permitted; the string "MCS01" will not be accepted. For 11ac rate, value is required to be a string with format "MCSx0y", where x means Nss (number of spatial streams) and y means MCS index, the possible value are 100 $\sim$ 109, 200 $\sim$ 209, 300 $\sim$ 309, 400 $\sim$ 409. This API cannot be used to configure auto rate fallback; use the Set Option API with qcsapi_autorate_fallback as the option to select auto rate fallback.

Note

This API can only be used on the primary interface (wifi0)

This API can only be used on the primary interface (wifi0)

To set an 802.11n MCS on a VHT capable device, you must first set the bandwidth to 20MHz or 40MHz.

See also

qcsapi_wifi_set_bw

Note

This API should only be used to evaluate the performance of a particular MCS (modulation and coding) index. Using it in a production application (i.e. with the end-user) can result in unexpectedly poor performance, either lower than expected transfer rates or a failure to associate. Use of the auto rate fallback option is strongly recommended.

If option autorate fallback is enabled, this API will disable it as a side effect.

Parameters

ifname	the primary WiFi interface, wifi0 only.
new_mcs_rate	the new MCS rate to use (fixed rate).

`call_qcsapi` interface:

`call_qcsapi set_mcs_rate <WiFi interface> <MCS index string>`

where <MCS index string> is an MCS index string.

See the description of the API itself for the expected format of the MCS index string.

Unless an error occurs, the output will be the string `complete`.

Note

This command cannot be used to configure the auto rate fallback option; use call_qcsapi set_option with autorate as the option for that purpose.

9.23.2.3 qcsapi_wifi_set_pairing_id()

```
int qcsapi_wifi_set_pairing_id (
            const char * ifname,
            const char * pairing_id )
```

Set pairing ID for use of pairing protection

The pairing ID is a 32 characters' string

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
pairing_id	a 32 characters' string used for pairing protection.

Returns

0 if the command succeeded and the pairing ID is updated.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_pairing_id <WiFi interface> <pairing ID>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.4 qcsapi_wifi_get_pairing_id()

```
int qcsapi_wifi_get_pairing_id (
            const char * ifname,
            char * pairing_id )
```

Get pairing ID which is for use of pairing protection

The pairing ID is a 32 characters' string

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
pairing_id	a 32 characters' string used for pairing protection.

Returns

0 if the pairing ID is fetched.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_pairing_id <WiFi interface>
```

Unless an error occurs, the output will be the string of pairing ID.

9.23.2.5 qcsapi_wifi_set_pairing_enable()

```
int qcsapi_wifi_set_pairing_enable (
            const char * ifname,
            const char * enable )
```

enable/disable the pairing protection

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable	Enabling mode of the pairing protection. 0 - disable 1 - enable and accept the association when pairing ID matches. 2 - enable and deby the association when pairing ID matches.

Returns

0 if the command succeeded and the pairing enable flag is updated.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_pairing_enable <WiFi interface> <pairing enable flag>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.6 qcsapi_wifi_get_pairing_enable()

```
int qcsapi_wifi_get_pairing_enable (
            const char * ifname,
            char * enable )
```

Get pairing enable flag which is for enabling pairing protection

Note

This API can only be used in AP mode.

The pairing enable flag is "0" or "1"

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable	a string used for enabling pairing protection.

Returns

0 if the enable flag is fetched.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_pairing_enable <WiFi interface>
```

Unless an error occurs, the output will be the string of enable flag

9.23.2.7 qcsapi_non_wps_set_pp_enable()

```
int qcsapi_non_wps_set_pp_enable (
            const char * ifname,
            uint32_t ctrl_state )
```

Set non_WPS pairing enable flag which is for enabling pairing protection

The pairing enable flag is "0" or "1"

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ctrl_state	a string used for enabling non WPS pairing protection.

Returns

0 if the enable flag is set.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_non_wps_pp_enable <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.8 qcsapi_non_wps_get_pp_enable()

```
int qcsapi_non_wps_get_pp_enable (
            const char * ifname,
            uint32_t * ctrl_state )
```

Get non_WPS pairing enable flag which is for enabling pairing protection

The pairing enable flag is "0" or "1"

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ctrl_state	a string used for getting the non WPS pairing protection status.

Returns

0 if the enable flag is fetched.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_non_wps_pp_enable <WiFi interface>
```

Unless an error occurs, the output will be the string of enable flag

9.23.2.9 qcsapi_wifi_set_vendor_fix()

```
int qcsapi_wifi_set_vendor_fix (
            const char * ifname,
            int fix_param,
            int value )
```

Set various fix items for compatibility issue with other vendor chipset.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
fix_param	the param to enable or disable the fix.
value	enable(1) or disable(0) the fix.

Returns

0 if the enabling or disabling the fix is successful.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_vendor_fix <WiFi interface> <fix-param> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.10 qcsapi_errno_get_message()

```
int qcsapi_errno_get_message (
            const int qcsapi_retval,
            char * error_msg,
            unsigned int msglen )
```

Given a numeric error code, convert to a human readable string. This wills for conventional negative errno values, as well as QCSAPI negative error values (<= -1000).

Parameters

qcsapi_retval	a negative error value to find the associated string of
error_msg	memory for result storage
msglen	length of *error_msg* buffer in bytes, including the null terminator

`call_qcsapi` interface:

```
call_qcsapi get_error_message <errno>
```

where <errno> is a negative error value

Output will be the requested error message, or the relevant error message if an error occurs.

9.23.2.11 qcsapi_wifi_get_vco_lock_detect_mode()

```
int qcsapi_wifi_get_vco_lock_detect_mode (
            const char * ifname,
            unsigned int * p_jedecid )
```

This API get the vco lock detect function is enabled or disabled.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be the status of lock detect function enabled .

\call_qcsapi

```
call_qcsapi get_vco_lock_detect_status <WiFi interface>
```

9.23.2.12 qcsapi_wifi_set_vco_lock_detect_mode()

```
int qcsapi_wifi_set_vco_lock_detect_mode (
            const char * ifname,
            unsigned int * p_jedecid )
```

This API set the vco lock detect function start/stop.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error.

Unless an error occurs, the output will be `1` (enabled) or `0` (disabled)

\call_qcsapi

```
call_qcsapi set_vco_lock_detect_mode <WiFi interface>
```

9.23.2.13 qcsapi_get_core_dump_size()

```
int qcsapi_get_core_dump_size (
            uint32_t * core_dump_size )
```

Parameters

core_dump_size	Size of the core dump (in bytes)

Returns

>=0 on success, <0 on error.

Unless an error occurs, the buf will contains the size of the core dump.

See also

qcsapi_get_core_dump

9.23.2.14 qcsapi_get_core_dump()

```
int qcsapi_get_core_dump (
            string_4096 buf,
            uint32_t bytes_to_copy,
            uint32_t start_offset,
            uint32_t * bytes_copied )
```

Parameters

buf	Pointer to the buffer that should contain the core dump.
bytes_to_copy	Maximum number of bytes that would be copied from the internal buffer. 'buf' should be atleast 'bytes_to_copy' bytes long.
start_offset	Offset into the internal buffer from where the API would start copying.
bytes_copied	Number of bytes actually copied from the internal buffer. If 'bytes_copied' < 'bytes_to_copy', there is no more data in the internal buffer or an error has occured.

Returns

>=0 on success, <0 on error.

Unless an error occurs, the buf will contains the requested dump..

Note

The client should first invoke qcsapi_get_core_dump_size(...) to get the size of the core dump. The dump can then be retrived in one shot (bytes_to_copy = core_dump_size, start_offset = 0) or in multiple chunks (bytes_to_copy = "chunk size", start_offset = "offset into the core dump" by invoking qcsapi_get_core_dump(...).

`call_qcsapi` interface:

```
call_qcsapi get_core_dump
```

Will output the core dump or a relevant error message if an error occurs.

Note

Since the core dump could contain non-ASCII characters, it is better to redirect the output of this command to a file

See also

qcsapi_get_core_dump_size

9.23.2.15 qcsapi_get_app_core_dump_size()

```
int qcsapi_get_app_core_dump_size (
            char * file,
            uint32_t * core_dump_size )
```

Parameters

file	Name of the core dump file
core_dump_size	Size of the core dump (in bytes)

Returns

>=0 on success, <0 on error

Unless an error occurs, the core_dump_size will hold the size of the core dump.

9.23.2.16 qcsapi_get_app_core_dump()

```
int qcsapi_get_app_core_dump (
            char * file,
            string_4096 buf,
            uint32_t bytes_to_copy,
            uint32_t offset,
            uint32_t * bytes_copied )
```

Parameters

file	Name of the core dump file to be retrieved
buf	Pointer to the buffer that should contain the core dump.
bytes_to_copy	Maximum number of bytes that would be copied from the internal buffer. 'buf' should be atleast 'bytes_to_copy' bytes long.
offset	Offset into the internal buffer from where the API would start copying.
bytes_copied	Number of bytes actually copied from the internal buffer. If 'bytes_copied' < 'bytes_to_copy', there is no more data in the internal buffer or an error has occured.

Returns

>=0 on success, <0 on error.

Unless an error occurs, the buf will contain the requested dump.

Note

The client should first invoke qcsapi_get_app_core_dump_size(...) to get the size of the core dump. The dump can then be retrived in one shot (bytes_to_copy = core_dump_size, offset = 0) or in multiple chunks (bytes↩ _to_copy = "chunk size", offset = "offset into the core dump" by invoking qcsapi_get_app_core_dump(...).

`call_qcsapi` interface:

```
call_qcsapi get_app_core_dump <core dump file> <output file>
```

Will output the core dump or a relevant error message if an error occurs.

See also

qcsapi_get_app_core_dump_size

9.23.2.17 qcsapi_set_log_level()

```
int qcsapi_set_log_level (
            const char * ifname,
            qcsapi_log_module_name index,
            const string_128 params )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
index	the module index. The module can be one of the values of the qcsapi_log_module_name enumeration.
params	module specific parameters (see below for valid values).

Returns

0 on success or negative values on error.

The supported params are as per below:

- Linux kernel: level=LEVEL LEVEL can be between 1 and 8.
- wpa_supplicant: level=LEVEL LEVEL can be ERROR, WARNING, INFO, DEBUG, MSGDUMP, EXCESSIVE.
- hostapd: level=LEVEL LEVEL can be ERROR, WARNING, INFO, DEBUG, MSGDUMP, EXCESSIVE. module=MODULE:level=LEVEL MODULE can be IEEE80211, IEEE8021X, RADIUS, WPA, DRIVER, IAPP, MLME. LEVEL can be VERBOSE, DEBUG, INFO, NOTICE, WARNING. level=LEVEL:module=MODULE:level=MODULE_LEVEL LEVEL can be ERROR, WARNING, INFO, DEBUG, MSGDUMP, EXCESSIVE. MODULE can be IEEE80211, IEEE8021X, RADIUS, WPA, DRIVER, IAPP, MLME. MODULE_LEVEL can be VERBOSE, DEBUG, INFO, NOTICE, WARNING.
- Driver: level=LEVEL LEVEL can be 0x00000000 to 0xffffffff.

`call_qcsapi` interface:

`call_qcsapi set_log_level <WiFi interface> <module name> <level=LEVEL>` For hostapd, you could also send the level for sub-modules as: `call_qcsapi set_log_level <WiFi interface> <module name> <level=LEVEL>` `call_qcsapi set_log_level <WiFi interface> <module name> <module=MODULE>:<level=MODULE_LEVEL>` `call_qcsapi set_log_level <WiFi interface> <module name> <level=LEVEL>:<module=MODULE>:<level=MODULE_LEVEL>`

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_get_log_level

qcsapi_log_module_name

9.23.2.18 qcsapi_get_log_level()

```
int qcsapi_get_log_level (
            const char * ifname,
            qcsapi_log_module_name index,
            string_128 params )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
index	the module index. The module can be one of the values of the qcsapi_log_module_name enumeration.
params	current log level set in the module (see below for valid values). For hostapd, Current level, module and module level (see below for valid values).

Returns

0 on success or negative values on error.

The supported params are as per below:

- Linux kernel: level=LEVEL LEVEL can be between 1 and 8.
- wpa_supplicant: level=LEVEL LEVEL can be ERROR, WARNING, INFO, DEBUG, MSGDUMP, EXCESSIVE.
- hostapd: level=LEVEL:module=MODULE:level=MODULE_LEVEL LEVEL can be ERROR, WARNING, INFO, DEBUG, MSGDUMP, EXCESSIVE. MODULE can be IEEE80211, IEEE8021X, RADIUS, WPA, DRIVER, IAPP, MLME. MODULE_LEVEL can be VERBOSE, DEBUG, INFO, NOTICE, WARNING.
- Driver: level=LEVEL LEVEL can be 0x00000000 to 0xffffffff.

`call_qcsapi` interface:

```
call_qcsapi get_log_level <WiFi interface> <module name>
```

Unless an error occurs, the output will be the log level configured for the given module.

See also

qcsapi_set_log_level

qcsapi_log_module_name

9.23.2.19 qcsapi_set_remote_logging()

```
int qcsapi_set_remote_logging (
            qcsapi_remote_log_action action_type,
            qcsapi_unsigned_int ipaddr )
```

Parameters

action_type	indicates one of the actions defined in enum qcsapi_remote_log_action
ipaddr	NPU's IP address where the logs would be streamed

Returns

0 on success or negative values on error.

`call_qcsapi` interface:

```
call_qcsapi set_remote_logging <enable> <ipaddr>  call_qcsapi set_remote_↩
logging <disable>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_remote_log_action

9.23.2.20 qcsapi_set_console()

```
int qcsapi_set_console (
            qcsapi_console_action action_type )
```

Parameters

action_type	indicates one of the actions defined in enum qcsapi_console_action

Returns

0 on success or negative values on error.

`call_qcsapi` interface:

```
call_qcsapi set_console <enable> call_qcsapi set_console <disable>
```

Unless an error occurs, the output will be the string `complete`.

See also

qcsapi_console_action

9.23.2.21 qcsapi_do_system_action()

```
int qcsapi_do_system_action (
            const string_32 action )
```

This API is used to perform the specified system action. And send the events to qevt_server.

Parameters

action	is of type string, specifies the system action to be performed.

Supported actions include:

- powerdown
- powerup
- reboot

```
call_qcsapi do_system_action <action>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.22 qcsapi_wifi_set_max_boot_cac_duration()

```
int qcsapi_wifi_set_max_boot_cac_duration (
            const char * ifname,
            const int max_boot_cac_duration )
```

This API call is used to configure the max cac duration to be used at boot time.

Note

This API can only be used in AP mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
max_boot_cac_duration	Maximum boot time CAC duration in seconds. The valid range is [(-1) - MAX_BOOT_CAC_DURATION].

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi set_max_boot_cac_duration <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.23 qcsapi_wifi_set_br_isolate()

```
int qcsapi_wifi_set_br_isolate (
            qcsapi_br_isolate_cmd cmd,
            uint32_t arg )
```

This API call is used to configure Quantenna bridge isolation feature.

Parameters

cmd	e_qcsapi_br_isolate_normal or e_qcsapi_br_isolate_vlan
arg	if cmd is e_qcsapi_br_isolate_normal: 0 (disable) or 1 (enable)
arg	if cmd is e_qcsapi_br_isolate_vlan: 0: disable VLAN br isolation 1-4095: enable VLAN br isolation for the specified VLAN 65535: enable VLAN br isolation for all VLAN packets

Returns

>=0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi set_br_isolate normal { 0 | 1 }
```
or
```
call_qcsapi set_br_isolate vlan {all | none | <VLAN ID>}
```

Unless an error occurs, the output will be the string `complete`.

9.23.2.24 qcsapi_wifi_get_br_isolate()

```
int qcsapi_wifi_get_br_isolate (
            uint32_t * result )
```

This API call is used to get Quantenna bridge isolation configuration.

Parameters

result	receive bridge isolation configuration from underlying driver: bit 0 – indicate whether normal bridge isolation is enabled bit 1 – indicate whether VLAN bridge isolation is enabled bit 15-31 – 0: VLAN bridge isolation enabled and all VLAN packets are isolated 1-4094: VLAN bridge isolation enabled and specified VLAN packets are isolated

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_br_isolate
```

The output will be the current bridge isolation configuration.

9.24 Video Stream Protection APIs

VSP ensures that video streams transmitted through Quantenna Access Points are protected from interference by other intra-BSS traffic and channel interference. VSP pro-actively monitors and adjusts the streams running across the network to ensure video streams do not drop or suffer artefacts. These APIs provide configuration for the VSP module.

Data Structures

- struct qvsp_hash_flds_ipv4
- struct qvsp_hash_flds_ipv6
- union qvsp_hash_flds
- struct qvsp_wl_flds
- struct qvsp_rule_flds
- struct qvsp_strm_stats
- struct qvsp_stats_if
- struct qvsp_stats
- struct qvsp_strm_info
- struct qvsp_strm_info_safe
- struct qvsp_strms

Macros

- #define **QVSP_STRM_MAX_ENTRIES** 256

9.24.1 Detailed Description

9.24.2 VSP Configuration options

- 0: QVSP_CFG_ENABLED - enabled
  Default: 0 number [0 - 1]. QTM enabled
- 1: QVSP_CFG_ENABLED_ALWAYS - enabled_always
  Default: 0 number [0 - 1]. QTM enabled when no QTM peers
- 2: QVSP_CFG_FAT_MIN - fat_min
  Default: 100 msps [1 - 1000]. Min free airtime
- 3: QVSP_CFG_FAT_MIN_SOFT - fat_min_soft
  Default: 170 msps [1 - 1000]. Soft min free airtime
- 4: QVSP_CFG_FAT_MIN_SOFT_CONSEC - fat_min_soft_consec
  Default: 3 number [1 - 255]. Consecutive soft min free airtime
- 5: QVSP_CFG_FAT_MIN_SAFE - fat_min_safe
  Default: 200 msps [1 - 1000]. Safe min free airtime
- 6: QVSP_CFG_FAT_MIN_CHECK_INTV - fat_min_check_intv
  Default: 2000 ms [100 - 60000]. Oversubscription check interval
- 7: QVSP_CFG_FAT_MAX_SOFT - fat_max_soft
  Default: 350 msps [1 - 1000]. Soft max free airtime

- 8: QVSP_CFG_FAT_MAX_SOFT_CONSEC - fat_max_soft_consec
  Default: 5 number [1 - 255]. Consecutive soft max free airtime
- 9: QVSP_CFG_FAT_MAX_SAFE - fat_max_safe
  Default: 250 msps [1 - 1000]. Safe max free airtime
- 10: QVSP_CFG_FAT_MAX_CHECK_INTV - fat_max_check_intv
  Default: 2000 ms [100 - 86400000]. Undersubscription check interval
- 11: QVSP_CFG_NODE_DATA_MIN - node_data_min
  Default: 480 Kbps [1 - 10000]. Min data for node cost update
- 12: QVSP_CFG_DISABLE_DEMOTE - disable_demote
  Default: 1 number [0 - 1]. Demote stream to disable
- 13: QVSP_CFG_DISABLE_DEMOTE_FIX_FAT - disable_demote_fat_fix
  Default: 0 number [0 - 1]. Adjust FAT when demoting streams
- 14: QVSP_CFG_DISABLE_WAIT - disable_wait
  Default: 3 secs [1 - 86400]. Min re-disable wait time
- 15: QVSP_CFG_DISABLE_PER_EVENT_MAX - disable_event_max
  Default: 1 number [1 - 256]. Max streams disabled per event
- 16: QVSP_CFG_ENABLE_WAIT - enable_wait
  Default: 15 secs [1 - 86400]. Min re-enable wait time
- 17: QVSP_CFG_ENABLE_PER_EVENT_MAX - enable_event_max
  Default: 1 number [1 - 256]. Max streams enabled per event
- 18: QVSP_CFG_STRM_RMT_DIS_TCP - rmt_disable_tcp
  Default: 1 number [0 - 1]. Disable Rx TCP streams at STA
- 19: QVSP_CFG_STRM_RMT_DIS_UDP - rmt_disable_udp
  Default: 1 number [0 - 1]. Disable Rx UDP streams at STA
- 20: QVSP_CFG_STRM_TPUT_MIN - strm_tput_min
  Default: 1000 Kbps [1 - 10000]. Min throughput for a real stream
- 21: QVSP_CFG_STRM_DISABLED_MAX - strm_disabled_max
  Default: 500 Kbps [20 - 10000]. Max throughput when disabled
- 22: QVSP_CFG_STRM_ADPT_THROT - strm_adpt_throt
  Default: 1 number [0 - 1]. Adaptive throttling enabled
- 23: QVSP_CFG_STRM_ADPT_THROT_STEP - strm_adpt_throt_step
  Default: 40 percent [1 - 100]. Adaptive throttling cost step
- 24: QVSP_CFG_STRM_ADPT_THROT_MARGIN - strm_adpt_throt_margin
  Default: 10000 Kbps [0 - 100000]. Adaptive throttling margin
- 25: QVSP_CFG_STRM_TPUT_SMPL_MIN - strm_tput_smpl_min
  Default: 20 ms [1 - 1000]. Min throughput sampling ms
- 26: QVSP_CFG_STRM_COST_RC_ADJUST - strm_cost_rc_adjust
  Default: 1 number [0 - 1]. Adjust stream cost for rate change
- 27: QVSP_CFG_STRM_MAX - strm_max
  Default: 256 cnt [1 - 256]. Max streams
- 28: QVSP_CFG_STRM_MAX_AC0 - strm_max_ac0
  Default: 0 cnt [0 - 256]. Max streams for AC 0
- 29: QVSP_CFG_STRM_MAX_AC1 - strm_max_ac1
  Default: 0 cnt [0 - 256]. Max streams for AC 1

- 30: QVSP_CFG_STRM_MAX_AC2 - strm_max_ac2
  Default: 0 cnt [0 - 256]. Max streams for AC 2
- 31: QVSP_CFG_STRM_MAX_AC3 - strm_max_ac3
  Default: 0 cnt [0 - 256]. Max streams for AC 3
- 32: QVSP_CFG_STRM_MIN - strm_min
  Default: 1 cnt [1 - 1000]. Min streams
- 33: QVSP_CFG_STRM_MIN_AC0 - strm_min_ac0
  Default: 0 cnt [1 - 1000]. Min streams for AC 0
- 34: QVSP_CFG_STRM_MIN_AC1 - strm_min_ac1
  Default: 0 cnt [1 - 1000]. Min streams for AC 1
- 35: QVSP_CFG_STRM_MIN_AC2 - strm_min_ac2
  Default: 0 cnt [1 - 1000]. Min streams for AC 2
- 36: QVSP_CFG_STRM_MIN_AC3 - strm_min_ac3
  Default: 0 cnt [1 - 1000]. Min streams for AC 3
- 37: QVSP_CFG_STRM_TPUT_MAX_TCP - strm_tput_max_tcp
  Default: 0 Mbps [0 - 10000]. Max stream throughput for TCP
- 38: QVSP_CFG_STRM_TPUT_MAX_TCP_AC0 - strm_tput_max_tcp_ac0
  Default: 0 Mbps [0 - 10000]. Max stream throughput for TCP AC 0
- 39: QVSP_CFG_STRM_TPUT_MAX_TCP_AC1 - strm_tput_max_tcp_ac1
  Default: 0 Mbps [0 - 10000]. Max stream throughput for TCP AC 1
- 40: QVSP_CFG_STRM_TPUT_MAX_TCP_AC2 - strm_tput_max_tcp_ac2
  Default: 0 Mbps [0 - 10000]. Max stream throughput for TCP AC 2
- 41: QVSP_CFG_STRM_TPUT_MAX_TCP_AC3 - strm_tput_max_tcp_ac3
  Default: 0 Mbps [0 - 10000]. Max stream throughput for TCP AC 3
- 42: QVSP_CFG_STRM_TPUT_MAX_UDP - strm_tput_max_udp
  Default: 0 Mbps [0 - 10000]. Max stream throughput for UDP
- 43: QVSP_CFG_STRM_TPUT_MAX_UDP_AC0 - strm_tput_max_udp_ac0
  Default: 0 Mbps [0 - 10000]. Max stream throughput for UDP AC 0
- 44: QVSP_CFG_STRM_TPUT_MAX_UDP_AC1 - strm_tput_max_udp_ac1
  Default: 0 Mbps [0 - 10000]. Max stream throughput for UDP AC 1
- 45: QVSP_CFG_STRM_TPUT_MAX_UDP_AC2 - strm_tput_max_udp_ac2
  Default: 0 Mbps [0 - 10000]. Max stream throughput for UDP AC 2
- 46: QVSP_CFG_STRM_TPUT_MAX_UDP_AC3 - strm_tput_max_udp_ac3
  Default: 0 Mbps [0 - 10000]. Max stream throughput for UDP AC 3
- 47: QVSP_CFG_STRM_ENABLE_WAIT - strm_enable_wait
  Default: 30 secs [1 - 86400]. Min stream re-enable wait time
- 48: QVSP_CFG_STRM_AGE_MAX - strm_age_max
  Default: 5 secs [1 - 86400]. Max stream age
- 49: QVSP_CFG_AGE_CHK_INTV - age_check_intv
  Default: 10 secs [1 - 86400]. Age check interval
- 50: QVSP_CFG_3RDPT_CTL - 3rd_party_ctl
  Default: 0 number [0 - 1]. Enable 3rd party client control
- 51: QVSP_CFG_3RDPT_LOCAL_THROT - 3rd_party_local_throt
  Default: 0 number [0 - 1]. Throttling 3rd party client packet also in local

- 52: QVSP_CFG_3RDPT_QTN - 3rd_party_qtn
  Default: 0 number [0 - 1]. Treat qtn client as 3rd party client
- 53: QVSP_CFG_BA_THROT_INTV - ba_throt_intv
  Default: 1000 ms [0 - 10000]. BA throttling interval
- 54: QVSP_CFG_BA_THROT_DUR_MIN - ba_throt_dur_min
  Default: 50 ms [0 - 10000]. BA throttling min duration
- 55: QVSP_CFG_BA_THROT_DUR_STEP - ba_throt_dur_step
  Default: 100 ms [50 - 10000]. BA throttling duration step
- 56: QVSP_CFG_BA_THROT_WINSIZE_MIN - ba_throt_winsize_min
  Default: 1 number [0 - 256]. BA throttling min winsize
- 57: QVSP_CFG_BA_THROT_WINSIZE_MAX - ba_throt_winsize_max
  Default: 16 number [1 - 256]. BA throttling max winsize
- 58: QVSP_CFG_WME_THROT_AC - wme_throt_ac
  Default: 3 number [0 - 15]. WME throttling AC bitmap
- 59: QVSP_CFG_WME_THROT_AIFSN - wme_throt_aifsn
  Default: 15 number [0 - 15]. WME throttling AIFSN
- 60: QVSP_CFG_WME_THROT_ECWMIN - wme_throt_ecwmin
  Default: 14 number [1 - 14]. WME throttling encoded cwmin
- 61: QVSP_CFG_WME_THROT_ECWMAX - wme_throt_ecwmax
  Default: 15 number [1 - 15]. WME throttling encoded cwmax
- 62: QVSP_CFG_WME_THROT_TXOPLIMIT - wme_throt_txoplimit
  Default: 0 number [0 - 65535]. WME throttling TXOP limit
- 63: QVSP_CFG_WME_THROT_THRSH_DISABLED - wme_throt_thrsh_disabled
  Default: 150 number [0 - 1000]. WME throttling disabled stream cost threshold
- 64: QVSP_CFG_WME_THROT_THRSH_VICTIM - wme_throt_thrsh_victim
  Default: 150 number [0 - 1000]. WME throttling victim stream cost threshold
- 65: QVSP_CFG_EVENT_LOG_LVL - event_level
  Default: 0 number [0 - 9]. Event log level
- 66: QVSP_CFG_DEBUG_LOG_LVL - debug_level
  Default: 3 number [0 - 9]. Debug log level

9.24.3 VSP Rule options

- 0: QVSP_RULE_PARAM_DIR - dir
  [0 - 2 val]. Direction
  Possible values are:
  - QVSP_RULE_DIR_ANY - Any
  - QVSP_RULE_DIR_TX - Tx
  - QVSP_RULE_DIR_RX - Rx
- 1: QVSP_RULE_PARAM_VAPPRI - vappri
  [1 - 15 bitmap]. VAP Priority
  Possible values are:
  - 0x01 = VAP priority 0
  - 0x02 = VAP priority 1

  - 0x04 = VAP priority 2
  - 0x08 = VAP priority 3

- 2: QVSP_RULE_PARAM_AC - ac
  [1 - 15 bitmap]. Access Classes
  Possible values are:
  - 0x01 = Best Effort (0)
  - 0x02 = Background (1)
  - 0x04 = Voice (2)
  - 0x08 = Video (3)
- 3: QVSP_RULE_PARAM_PROTOCOL - protocol
  [6 - 17 val]. IP protocol - TCP(6) or UDP(17)
- 4: QVSP_RULE_PARAM_TPUT_MIN - tp_min
  [1 - 10000 Mbps]. Min throughput
- 5: QVSP_RULE_PARAM_TPUT_MAX - tp_max
  [1 - 10000 Mbps]. Max throughput
- 6: QVSP_RULE_PARAM_COST_MIN - cost_min
  [1 - 1000 msps]. Cost min
- 7: QVSP_RULE_PARAM_COST_MAX - cost_max
  [1 - 1000 msps]. Cost max
- 8: QVSP_RULE_PARAM_ORDER - order
  [0 - 9 val]. Match order
  Allowed match orderings are (see enum qvsp_rule_order_e):
  - 0: QVSP_RULE_ORDER_GREATEST_COST_NODE
    greatest cost node first
  - 1: QVSP_RULE_ORDER_LEAST_COST_NODE
    least cost node first
  - 2: QVSP_RULE_ORDER_GREATEST_NODE_INV_PHY_RATE
    greatest inverse PHY rate node first
  - 3: QVSP_RULE_ORDER_LEAST_NODE_INV_PHY_RATE
    least inverse PHY rate node first
  - 4: QVSP_RULE_ORDER_GREATEST_COST_STREAM
    greatest cost stream first
  - 5: QVSP_RULE_ORDER_LEAST_COST_STREAM
    least cost stream first
  - 6: QVSP_RULE_ORDER_NEWEST
    newest first
  - 7: QVSP_RULE_ORDER_OLDEST
    oldest first
  - 8: QVSP_RULE_ORDER_LOWEST_TPUT
    lowest throughput first
  - 9: QVSP_RULE_ORDER_HIGHEST_TPUT
    highest throughput first
- 9: QVSP_RULE_PARAM_THROT_POLICY - throt_policy
  [1 - 2 val]. Throttling policy - binary(1) or adaptive(2)
- 10: QVSP_RULE_PARAM_DEMOTE - demote
  [0 - 1 val]. Demote stream

9.25 API to call scripts for EMI testing and RF testing

This chapter describes APIs to call scripts on the board to run EMI testing and RF testing in calstate=1 mode.

Functions

- int qcsapi_wifi_run_script (const char *scriptname, const char *param)
  API of scripts for EMI testing and RF testing.

9.25.1 Detailed Description

9.25.2 Function Documentation

9.25.2.1 qcsapi_wifi_run_script()

```
int qcsapi_wifi_run_script (
            const char * scriptname,
            const char * param )
```

This function is used to call a script on the board. This API should be used when device is configured to calstate=1 mode. The following scripts are supported:

set_test_mode
This script is used to configure the packet type.
`set_test_mode <Channel> <Antenna> <MCS Level> <BW> <Size> <11n signal> <BF>`
Channel: channel number of the center frequency
Antenna: 127 - 4 chanis on, 113 - chain 1 on, 116 - chain3 on, 120 - chain 4 on.
MCS level: MCS# of packet transmitted
BW: 20 or 40 in MHz units.
Size: packet size in 100bytes units, it should be a number smaller than 40 and bigger than 0
11n signal: 1 - 11n, 0 - 11a.
BF: 0 default.

send_test_packet
Start to transmit packet. Please note that before calling this script, test mode should be set by script `set_test_mode`
`send_test_packet <number>`
number: How many(number*1000) packets will be sent.

stop_test_packet
stop sending packet.

set_tx_pow x
set the packet output power to xdBm where x can vary depending on the front end device.

send_cw_signal
Generate CW tone for different channels for frequency offset measurement.
`send_cw_signal <channel> <chain> <CW pattern> <tone> <sideband>`
channel: channel number like 36, 40, 44, ...
chain: the value are 2 separate numbers.

- 0 0 - chain1
- 0 2 - chain2
- 1 0 - chain3
- 1 2 - chain4

CW pattern:

- 0 - 625KHz with 0 dBFS power
- 1 - 625KHz with -3 dBFS power
- 2 - 1MHz with 0 dBFS power
- 3 - 1MHz with -3dBFS power

stop_cw_signal
stop the CW tone.

send_cw_signal_4chain
Generate CW tone for different channels and send signal using all four chains, or stop the CW tone.
`send_cw_signal_4chain { start <channel> | stop }`

show_test_packet
show information about test packet.

Parameters

scriptname	the name of the script file
param	parameters used by the script

Returns

0 on success, negative on error.

`call_qcsapi` interface:

```
call_qcsapi run_script <scriptname> <parameters>
```

The output will be silent on success, or an error message on failure.

Example:
`call_qcsapi run_script set_test_mode 36 127 15 20 40 1 0`

9.26 API for PHY testing

These APIs are used for PHY testing.

Functions

- int qcsapi_calcmd_set_test_mode (qcsapi_unsigned_int channel, qcsapi_unsigned_int antenna, qcsapi_unsigned_int mcs, qcsapi_unsigned_int bw, qcsapi_unsigned_int pkt_size, qcsapi_unsigned_int eleven_n, qcsapi_unsigned_int bf)

  QCSAPI for calcmd SET_TEST_MODE.
- int qcsapi_calcmd_show_test_packet (qcsapi_unsigned_int *tx_packet_num, qcsapi_unsigned_int *rx_packet_num, qcsapi_unsigned_int *crc_packet_num)

  Show TX packet number, RX packet number and CRC number.
- int qcsapi_calcmd_send_test_packet (qcsapi_unsigned_int to_transmit_packet_num)

  Start sending OFDM Packet.
- int qcsapi_calcmd_stop_test_packet (void)

  Stop transmitting OFDM Packet.
- int qcsapi_calcmd_send_dc_cw_signal (qcsapi_unsigned_int channel)

  send CW signal at center frequency for Frequency offset measurement
- int qcsapi_calcmd_stop_dc_cw_signal (void)

  stop transmitting CW signal
- int qcsapi_calcmd_get_test_mode_antenna_sel (qcsapi_unsigned_int *antenna_bit_mask)

  get antenna selection
- int qcsapi_calcmd_get_test_mode_mcs (qcsapi_unsigned_int *test_mode_mcs)

  get mcs config
- int qcsapi_calcmd_get_test_mode_bw (qcsapi_unsigned_int *test_mode_bw)

  get bandwidth config
- int qcsapi_calcmd_get_tx_power (qcsapi_calcmd_tx_power_rsp *tx_power)

  get tx power value
- int qcsapi_calcmd_set_tx_power (qcsapi_unsigned_int tx_power)

  set target tx power
- int qcsapi_calcmd_get_test_mode_rssi (qcsapi_calcmd_rssi_rsp *test_mode_rssi)

  get RSSI value
- int qcsapi_calcmd_set_mac_filter (int q_num, int sec_enable, const qcsapi_mac_addr mac_addr)

  set mac filter
- int qcsapi_calcmd_get_antenna_count (qcsapi_unsigned_int *antenna_count)

  get number of antenna
- int qcsapi_calcmd_clear_counter (void)

  clear tx/rx counter
- int qcsapi_calcmd_get_info (string_1024 output_info)

  get firmware info

9.26.1 Detailed Description

9.26.2 Function Documentation

9.26.2.1 qcsapi_calcmd_set_test_mode()

```
int qcsapi_calcmd_set_test_mode (
            qcsapi_unsigned_int channel,
            qcsapi_unsigned_int antenna,
            qcsapi_unsigned_int mcs,
            qcsapi_unsigned_int bw,
            qcsapi_unsigned_int pkt_size,
            qcsapi_unsigned_int eleven_n,
            qcsapi_unsigned_int bf )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	value to set. The param belonging to each value is predefined.

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi set_test_mode calcmd <value:channel> <value:antenna> <value↩
:MCS> <value:BW> <value:Packet size> <value:11N> <value:beamforming>
```

Output is silent with a return code of zero if successful, and an error message on failure.

Example:
`call_qcsapi set_test_mode calcmd 36 127 15 40 40 1 0 (Channel 36, 4 antenna, MCS 15, HT40, 4Kbytes, 11N Signal, bf)`

9.26.2.2 qcsapi_calcmd_show_test_packet()

```
int qcsapi_calcmd_show_test_packet (
            qcsapi_unsigned_int * tx_packet_num,
            qcsapi_unsigned_int * rx_packet_num,
            qcsapi_unsigned_int * crc_packet_num )
```

This API call is used to show test mode packet statistics.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
stats	return parameter to contain the statistics of TX, RX, CRC packet number.

Returns

>= 0 on success, < 0 on error. If success, stats contains the packet number statistics data retrieved from the device for the interface.

`call_qcsapi` interface:

```
call_qcsapi show_test_packet calcmd
```

Unless an error occurs, the output will be the PHY statistics data of the interface.

9.26.2.3 qcsapi_calcmd_send_test_packet()

```
int qcsapi_calcmd_send_test_packet (
            qcsapi_unsigned_int to_transmit_packet_num )
```

This API call is used to send OFDM packet.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	the number of packet to transmit, 1 is 1000 packets. If number is set, it will automatically stop, if 0, transmit infinitely

Returns

>= 0 on success, < 0 on error. If success, stats contains the packet number statistics data retrieved from the device for the interface.

`call_qcsapi` interface:

```
call_qcsapi send_test_packet calcmd <value>
```

Output is silent with a return code of zero if successful, and an error message on failure.

9.26.2.4 qcsapi_calcmd_stop_test_packet()

```
int qcsapi_calcmd_stop_test_packet (
            void  )
```

This API call is used to stop transmitting OFDM packet in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi stop_test_packet calcmd
```

Output is silent with a return code of zero if successful, and an error message on failure.

9.26.2.5 qcsapi_calcmd_send_dc_cw_signal()

```
int qcsapi_calcmd_send_dc_cw_signal (
            qcsapi_unsigned_int channel )
```

This API call is used to send continuous signal in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
channel	to perform the action on. (calcmd)

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi send_dc_cw_signal calcmd <channel>
```

Output is silent with a return code of zero if successful, and an error message on failure.

9.26.2.6 qcsapi_calcmd_stop_dc_cw_signal()

```
int qcsapi_calcmd_stop_dc_cw_signal (
            void  )
```

This API call is used to stop Continuous signal in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi stop_dc_cw_signal calcmd
```

Output is silent with a return code of zero if successful, and an error message on failure.

9.26.2.7 qcsapi_calcmd_get_test_mode_antenna_sel()

```
int qcsapi_calcmd_get_test_mode_antenna_sel (
            qcsapi_unsigned_int * antenna_bit_mask )
```

This API call is used to retrieve antenna configuration in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi get_test_mode_antenna_sel calcmd
```

Output is antenna number with bit mask type. e.g) 1010 means Ant4 & Ant2 is enabled.

9.26.2.8 qcsapi_calcmd_get_test_mode_mcs()

```
int qcsapi_calcmd_get_test_mode_mcs (
            qcsapi_unsigned_int * test_mode_mcs )
```

This API call is used to retrieve MCS configuration in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi get_test_mode_mcs calcmd
```

Output is MCS configuration.

9.26.2.9 qcsapi_calcmd_get_test_mode_bw()

```
int qcsapi_calcmd_get_test_mode_bw (
            qcsapi_unsigned_int * test_mode_bw )
```

This API call is used to retrieve bandwidth configuration in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi get_test_mode_bw calcmd
```

Output is Bandwidth configuration.

9.26.2.10 qcsapi_calcmd_get_tx_power()

```
int qcsapi_calcmd_get_tx_power (
            qcsapi_calcmd_tx_power_rsp * tx_power )
```

This API call is used to retrieve current TX power.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi get_test_mode_tx_power calcmd
```

Output is TX power value.

9.26.2.11 qcsapi_calcmd_set_tx_power()

```
int qcsapi_calcmd_set_tx_power (
            qcsapi_unsigned_int tx_power )
```

This API call is used to set TX power.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

\value target power to transmission

Returns

>= 0 on success, < 0 on error. If success, stats contains

call_qcsapi interface:

```
call_qcsapi set_test_mode_tx_power calcmd <value>
```

9.26.2.12 qcsapi_calcmd_get_test_mode_rssi()

```
int qcsapi_calcmd_get_test_mode_rssi (
            qcsapi_calcmd_rssi_rsp * test_mode_rssi )
```

This API call is used to retrieve RSSI value in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

call_qcsapi interface:

```
call_qcsapi get_test_mode_rssi calcmd
```

Output is RSSI value.

9.26.2.13 qcsapi_calcmd_set_mac_filter()

```
int qcsapi_calcmd_set_mac_filter (
            int q_num,
            int sec_enable,
            const qcsapi_mac_addr mac_addr )
```

This API call is used to set mac filter.

Parameters

q_num	the contention queue number
sec_enable	the security if enable or not
mac_addr	the mac address which the device is used to filter the packet

Returns

>= 0 on success, < 0 on error. If success, stats contains

call_qcsapi interface:

```
call_qcsapi calcmd_set_mac_filter calcmd <q_num> <sec_enable> <mac_addr>
```

Unless an error occurs, the output will be the string `complete.`

Example:
`call_qcsapi calcmd_set_mac_filter calcmd 0 2 00:11:22:33:44:55`

9.26.2.14 qcsapi_calcmd_get_antenna_count()

```
int qcsapi_calcmd_get_antenna_count (
            qcsapi_unsigned_int * antenna_count )
```

This API call is used to retrieve auntenna counter in test mode.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi get_antenna_counter calcmd
```

Output is number of antenna.

9.26.2.15 qcsapi_calcmd_clear_counter()

```
int qcsapi_calcmd_clear_counter (
            void  )
```

This API call is used to clear counter of tx/rx packets.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

Returns

>= 0 on success, < 0 on error. If success, stats contains

`call_qcsapi` interface:

```
call_qcsapi calcmd_clear_counter calcmd
```

9.26.2.16 qcsapi_calcmd_get_info()

```
int qcsapi_calcmd_get_info (
            string_1024 output_info )
```

This API call is used to retrieve firmware information.

Parameters

output_info	

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_info wifi0
```

Output is firmware info.

9.27 Bootcfg APIs

These APIs deal with the bootcfg interface.

Functions

- int qcsapi_bootcfg_get_parameter (const char *param_name, char *param_value, const size_t max_↩ param_len)

  Retrieve a parameter from bootcfg environment.
- int qcsapi_bootcfg_update_parameter (const char *param_name, const char *param_value)

  Persist a parameter in bootcfg environment flash.
- int qcsapi_bootcfg_commit (void)

  Sync bootcfg updates to flash.

9.27.1 Detailed Description

These APIs deal with the low level, early boottime configuration of the board.

Note

The update bootcfg API (qcsapi_bootcfg_update_parameter) can only be called in bringup mode, NOT in production mode.

9.27.2 Function Documentation

9.27.2.1 qcsapi_bootcfg_get_parameter()

```
int qcsapi_bootcfg_get_parameter (
            const char * param_name,
            char * param_value,
            const size_t max_param_len )
```

Read a u-boot environment parameter from the bootcfg driver, which manages persistent u-boot environment variables.

Parameters

param_name	Name of parameter being requested
param_value	Result storage for the value of the parameter
max_param_len	size of the buffer passed in *param_value*

Returns

0 on success, -ENODATA if the parameter is not found, or other negative error codes on failure.

`call_qcsapi` interface:

```
call_qcsapi get_bootcfg_param <param_name>
```

Output will be the value of the requested environment variable on success, or an error message on failure.

9.27.2.2 qcsapi_bootcfg_update_parameter()

```
int qcsapi_bootcfg_update_parameter (
            const char * param_name,
            const char * param_value )
```

Write a u-boot environment parameter to the bootcfg driver. Bootcfg driver will handle writing the new parameter to persistent storage.

Parameters

param_name	Name of parameter being set
param_value	Value of parameter to be set

Returns

0 on success, negative error codes on failure.

`call_qcsapi` interface:

```
call_qcsapi update_bootcfg_param <param_name> <param_value>
```

Unless an error occurs, the output will be the string `complete`.

9.27.2.3 qcsapi_bootcfg_commit()

```
int qcsapi_bootcfg_commit (
            void  )
```

This function can be called after making changes to bootcfg with `qcsapi_bootcfg_update_parameter`, to ensure that all pending updates have been committed to flash.

Note

This call will block until the flash has been written back. Generally this call will complete immediately with interactive use of call_qcsapi, and is used during production for ensuring scripts complete write of the bootcfg parameters prior to calling board reboot.

Returns

0 when all pending updates have been committed to flash

negative error codes on failure

`call_qcsapi` interface:

```
call_qcsapi commit_bootcfg
```

Unless an error occurs, the output will be the string `complete`.

9.28 Firmware Management APIs

These APIs are used for firmware management functions.

Macros

- #define **UBOOT_INFO_VER** 0
- #define **UBOOT_INFO_BUILT** 1
- #define **UBOOT_INFO_TYPE** 2
- #define **UBOOT_INFO_ALL** 3
- #define **UBOOT_INFO_LARGE** 0
- #define **UBOOT_INFO_MINI** 1
- #define **UBOOT_INFO_TINY** 2
- #define **MTD_DEV_BLOCK0** "/dev/mtdblock0"
- #define **MTD_UBOOT_VER_OFFSET** 11
- #define **MTD_UBOOT_OTHER_INFO_OFFSET** 32

Functions

- int qcsapi_get_uboot_info (string_32 uboot_version, struct early_flash_config ∗ef_config)

  Get u-boot information.
- int qcsapi_firmware_get_version (char ∗firmware_version, const qcsapi_unsigned_int version_size)

  Get the version of the firmware running on the device.
- int qcsapi_flash_image_update (const char ∗image_file, qcsapi_flash_partiton_type partition_to_upgrade)

  Update an image partition with the requested image.
- int qcsapi_send_file (const char ∗image_file_path, const int image_flags)

  Copy an image file from RC to EP.

9.28.1 Detailed Description

9.28.2 Function Documentation

9.28.2.1 qcsapi_get_uboot_info()

```
int qcsapi_get_uboot_info (
            string_32 uboot_version,
            struct early_flash_config * ef_config )
```

Parameters

uboot_info	(0 - ver, 1 - built, 2 - type, 3 - all) type can be U-boot (Mini) or U-boot (Large)

Returns

0 upon success, otherwise error value See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi get_uboot_info <uboot info type>
```

\usage call_qcsapi get_uboot_info 0 Version: v36.7.0.2

call_qcsapi get_uboot_info 1 Built: 05 June 2014 06:30:07

call_qcsapi get_uboot_info 2 Type: U-boot (Mini)

call_qcsapi get_uboot_info 3 Version: v36.7.0.2 Built : 05 June 2014 06:30:07 Type : U-boot (Mini)

9.28.2.2 qcsapi_firmware_get_version()

```
int qcsapi_firmware_get_version (
            char * firmware_version,
            const qcsapi_unsigned_int version_size )
```

This API reports the version of the firmware running on the device.

Parameters

firmware_version	return parameter string to contain the firmware version.
version_size	the size of the buffer firmware_version.

`call_qcsapi` interface:

```
call_qcsapi get_firmware_version
```

Unless an error occurs, the output will be the version of the firmware currently running on the device. It is not necessary to specify the version size parameter; it will default to 40 (characters).

9.28.2.3 qcsapi_flash_image_update()

```
int qcsapi_flash_image_update (
            const char * image_file,
            qcsapi_flash_partiton_type partition_to_upgrade )
```

This API updates either the live or safety partition with a new image. The image is checked to be for the appropriate architecture and checksummed before writing to flash.

Parameters

image_file	path to the new firmware image in the filesystem
partition_to_upgrade	either the live or safety partition

`call_qcsapi` interface:

```
call_qcsapi flash_image_update <path> {live|safety|uboot_live}
```

Unless an error occurs, the output will be the string `complete`.

Note

when used via RPC, timeout is 60 seconds

9.28.2.4 qcsapi_send_file()

```
int qcsapi_send_file (
            const char * image_file_path,
            const int image_flags )
```

This API transfers either kernel or bootloader image file from RC to /tmp directory of EP

Parameters

image_file	path to the new firmware image in the filesystem
image_flags	any additional flags, for future extensions

`call_qcsapi` interface:

```
call_qcsapi flash_image_update <path> <flags>
```

Unless an error occurs, the output will be the string `complete`. Default binding interface for the server is pcie0. This can be overwritten in /mnt/jffs2/qfts.conf file.

9.29 Power Management APIs

These APIs are used for power management functions.

Macros

- #define **QCSAPI_PM_MODE_DISABLE** BOARD_PM_LEVEL_FORCE_NO
- #define **QCSAPI_PM_MODE_AUTO** -1
- #define **QCSAPI_PM_MODE_IDLE** BOARD_PM_LEVEL_IDLE
- #define **QCSAPI_PM_MODE_SUSPEND** BOARD_PM_LEVEL_SUSPEND

Functions

- int qcsapi_set_aspm_l1 (int enable, int latency)

  enable/disable L1 state for the ASPM of PCIE.
- int qcsapi_set_l1 (int enter)

  enter/exit L1 state of PCIE link.
- int qcsapi_pm_set_mode (int mode)

  Set power save setting.
- int qcsapi_pm_dual_emac_set_mode (int mode)

  Set power save setting for Dual Ethernet.
- int qcsapi_pm_get_mode (int *mode)

  Get power save setting.
- int qcsapi_pm_dual_emac_get_mode (int *mode)

  Get power save setting for Daul Ethernet.
- int qcsapi_get_qpm_level (int *qpm_level)

  Get qpm level.
- int qcsapi_set_host_state (const char *ifname, const uint32_t host_state)

  set host state

9.29.1 Detailed Description

9.29.2 Function Documentation

9.29.2.1 qcsapi_set_aspm_l1()

```
int qcsapi_set_aspm_l1 (
            int enable,
            int latency )
```

when enabling the L1 state, the latency time can be set with parameter(0 ∼ 6) for L1 entry. 0 - 1us 1 - 2us 2 - 4us 3 - 8us 4 - 16us 5 - 32us 6 - 64us

Parameters

enable	0/1
latency	- time for L1 entry

Returns

0 if success.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi set_aspm_l1 <enable> <latency>
```

See QCSAPI Return Values for error codes and messages.

9.29.2.2 qcsapi_set_l1()

```
int qcsapi_set_l1 (
            int enter )
```

Parameters

enter	0/1

Returns

0 if success.

A negative value if an error occurred.

`call_qcsapi` interface:

```
call_qcsapi set_l1 <enter>
```

See QCSAPI Return Values for error codes and messages.

9.29.2.3 qcsapi_pm_set_mode()

```
int qcsapi_pm_set_mode (
            int mode )
```

Set the current power save setting

Parameters

mode	new power save setting. Valid values are: • *QCSAPI_PM_MODE_DISABLE* - Disable all power saving features • *QCSAPI_PM_MODE_AUTO* - Automatic; power saving will adjust itself based on associated stations, traffic levels etc • *QCSAPI_PM_MODE_IDLE* - Force to idle state • *QCSAPI_PM_MODE_SUSPEND* - Suspend all operations

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

The 'on' and 'auto' keywords are synonymous.

`call_qcsapi` interface:

```
call_qcsapi pm { off | on | auto | idle | suspend }
```

9.29.2.4 qcsapi_pm_dual_emac_set_mode()

```
int qcsapi_pm_dual_emac_set_mode (
            int mode )
```

Set the current power save setting for Ethernet ports, applied when two Ethernet ports are used

Parameters

mode	new power save setting. Valid values are: • *QCSAPI_PM_MODE_DISABLE* - Disable all power saving features • *QCSAPI_PM_MODE_AUTO* - Automatic; power saving will adjust itself based on associated stations, traffic levels etc

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

The 'on' and 'auto' keywords are synonymous.

`call_qcsapi` interface:

```
call_qcsapi pm { off | on | auto } dual_emac
```

Note

Dual Ethernet power saving is only activated when two Ethernet interfaces are up.

9.29.2.5 qcsapi_pm_get_mode()

```
int qcsapi_pm_get_mode (
            int * mode )
```

Get the current power save setting. This is related to the SoC power saving, not 802.11 power saving.

Parameters

mode	pointer to where the current power save setting value should be stored.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi pm
```

Unless an error occurs, the output will be the current value of power save setting from a list <off|auto|idle|suspend>.

9.29.2.6 qcsapi_pm_dual_emac_get_mode()

```
int qcsapi_pm_dual_emac_get_mode (
            int * mode )
```

Get the current power save setting. This is related to Dual Ethernet power saving.

Parameters

mode	pointer to where the current power save setting value should be stored.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi pm dual_emac
```

Unless an error occurs, the output will be the current value of power save setting from a list <off|auto|idle|suspend>.

Note

Dual Ethernet power saving is only activated when two Ethernet interfaces are up.

9.29.2.7 qcsapi_get_qpm_level()

```
int qcsapi_get_qpm_level (
            int * qpm_level )
```

Get the current qpm level. This is related to the SoC power saving

Parameters

qpm_level	to where the current qpm level value should be stored

Returns

0 if the command succeeded.

a negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi qpm_level
```

Unless an error occurs, the output will be a number in the range 0 to 6. Value 0 indicates power save disabled; Values 1 to 5 indicate the current power save level; Value 6 indicates power saving is suspended.

9.29.2.8 qcsapi_set_host_state()

```
int qcsapi_set_host_state (
            const char * ifname,
            const uint32_t host_state )
```

This API call is used for host CPU to inform radio module its state

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
host_state	either '0 (exit power save) or '1' (enter power save)

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi wowlan_host_state <WiFi interface> {0 | 1}
```

9.30 Smart Channel Select (SCS) APIs

These APIs are used to configure and get status for the Quantenna Smart Channel Select (SCS) feature.

Functions

- int qcsapi_wifi_get_scs_cce_channels (const char *ifname, qcsapi_unsigned_int *p_prev_channel, qcsapi_unsigned_int *p_cur_channel)

  Returns the channels during the last channel change event.
- int qcsapi_wifi_scs_enable (const char *ifname, uint16_t enable_val)

  Turn SCS feature on/off.
- int qcsapi_wifi_scs_switch_channel (const char *ifname, uint16_t pick_flags)

  Trigger SCS switch channel manually.
- int qcsapi_wifi_set_scs_verbose (const char *ifname, uint16_t enable_val)

  Turn SCS feature's verbose information output on/off.
- int qcsapi_wifi_get_scs_status (const char *ifname, qcsapi_unsigned_int *p_scs_status)

  Get the current enabled state of SCS feature.
- int qcsapi_wifi_set_scs_smpl_enable (const char *ifname, uint16_t enable_val)

  Turn SCS feature's channel sampling on/off.
- int qcsapi_wifi_set_scs_smpl_dwell_time (const char *ifname, uint16_t scs_sample_time)

  Set the duration for sampling the busyness of a channel.
- int qcsapi_wifi_set_scs_sample_intv (const char *ifname, uint16_t scs_sample_intv)

  Set the interval between two samples for SCS feature.
- int qcsapi_wifi_set_scs_intf_detect_intv (const char *ifname, uint16_t scs_intf_detect_intv)

  Set the interval between two interference detection for SCS feature.
- int qcsapi_wifi_set_scs_thrshld (const char *ifname, const char *scs_param_name, uint16_t scs_threshold)

  Set the threshold values for SCS feature.
- int qcsapi_wifi_set_scs_report_only (const char *ifname, uint16_t scs_report_only)

  Set if SCS feature should only report.
- int qcsapi_wifi_get_scs_stat_report (const char *ifname, struct qcsapi_scs_ranking_rpt *scs_rpt)

  Get the channel evaluation result for SCS feature.
- int qcsapi_wifi_get_scs_score_report (const char *ifname, struct qcsapi_scs_score_rpt *scs_rpt)

  Get the channel evaluation with scoring way for SCS feature.
- int qcsapi_wifi_get_scs_currchan_report (const char *ifname, struct qcsapi_scs_currchan_rpt *scs_currchan_rpt)

  Get current channel's stats for SCS feature.
- int qcsapi_wifi_set_scs_stats (const char *ifname, uint16_t start)

  Start/Stop SCS stats task.
- int qcsapi_wifi_get_autochan_report (const char *ifname, struct qcsapi_autochan_rpt *autochan_rpt)

  Get the initial auto channel evaluation result.
- int qcsapi_wifi_set_scs_cca_intf_smth_fctr (const char *ifname, uint8_t smth_fctr_noxp, uint8_t smth_fctr_xped)

  Set smoothing factor for SCS CCA interference measurement.
- int qcsapi_wifi_set_scs_chan_mtrc_mrgn (const char *ifname, uint8_t chan_mtrc_mrgn)

  Set channel metric margin for SCS channel ranking.
- int qcsapi_wifi_get_scs_cca_intf (const char *ifname, const qcsapi_unsigned_int the_channel, int *p_cca_intf)

  Get the specified channel's CCA interference level.
- int qcsapi_wifi_get_scs_param_report (const char *ifname, struct qcsapi_scs_param_rpt *p_scs_param_rpt, uint32_t param_num)

  Get the configured SCS parameters.
- int qcsapi_wifi_get_scs_dfs_reentry_request (const char *ifname, qcsapi_unsigned_int *p_scs_dfs_reentry_request)

  Get the current state of SCS DFS Re-entry request.

9.30.1 Detailed Description

9.30.2 Function Documentation

9.30.2.1 qcsapi_wifi_get_scs_cce_channels()

```
int qcsapi_wifi_get_scs_cce_channels (
            const char * ifname,
            qcsapi_unsigned_int * p_prev_channel,
            qcsapi_unsigned_int * p_cur_channel )
```

Retrieve the previous channel and the current channel during the last channel change event.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_prev_channel	the channel before the channel change.
p_cur_channel	the channel after the channel change.

Returns

0 on success or -EOPNOTSUPP or other negative values on error.

`call_qcsapi` interface:

```
call_qcsapi get_scs_cce_channels <WiFi interface>
```

Unless an error occurs, the previous and current channel will be displayed.

9.30.2.2 qcsapi_wifi_scs_enable()

```
int qcsapi_wifi_scs_enable (
            const char * ifname,
            uint16_t enable_val )
```

Turn the SCS feature on or off.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable_val	a value turn the feature on/off

Returns

0 on success or negative values on error

call_qcsapi interface:

```
call_qcsapi enable_scs <WiFi interface> [ 0 | 1 ]
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.3 qcsapi_wifi_scs_switch_channel()

```
int qcsapi_wifi_scs_switch_channel (
            const char * ifname,
            uint16_t pick_flags )
```

Trigger SCS switch channel manually, regardless whether there is interference or not, and whether the current channel is the best one or not.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).

pick_flags	flag used for channel selection Flags used as channel set are defined as below and they are mutually-exclusive:
	`IEEE80211_SCS_PICK_DFS_ONLY 0x1 IEEE80211_SCS_PICK_NON_DFS_ONLY 0x2`
	`IEEE80211_SCS_PICK_AVAILABLE_DFS_ONLY 0x4`
	`IEEE80211_SCS_PICK_AVAILABLE_ANY_CHANNEL 0x8`
	The header file `net80211/ieee80211_dfs_reentry.h` including this macros comes with the package libqcsapi_client_src.zip.

Where `<pick_flags>` should be "dfs", "non_dfs" or "all". This parameter indicates that what kind of channel to be selected. With using "dfs", It will pick channel from available dfs channels. With using "non-dfs", it will pick channel from available non-dfs channels. And "all" is default which means it will pick channel from all available channels.

Returns

0 on success or negative values on error

call_qcsapi interface:

```
call_qcsapi scs_switch_chan <WiFi interface> <pick flags>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.4 qcsapi_wifi_set_scs_verbose()

```
int qcsapi_wifi_set_scs_verbose (
            const char * ifname,
            uint16_t enable_val )
```

Turn the SCS feature's verbose information output on or off.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable_val	a value turn the verbose information output on/off

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_verbose <WiFi interface> [ 0 | 1 ]
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.5 qcsapi_wifi_get_scs_status()

```
int qcsapi_wifi_get_scs_status (
            const char * ifname,
            qcsapi_unsigned_int * p_scs_status )
```

Return the current enabled status of the SCS feature. It could be either enabled or disabled.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_scs_status	value that contains if SCS is enabled or disabled.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_status <WiFi interface>
```

The output will be the word "Disabled" or "Enabled" unless an error occurs.

9.30.2.6 qcsapi_wifi_set_scs_smpl_enable()

```
int qcsapi_wifi_set_scs_smpl_enable (
            const char * ifname,
            uint16_t enable_val )
```

Turn the SCS feature's channel sampling on or off.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable_val	a value turn the channel sampling on/off

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_smpl_enable <WiFi interface> [ 0 | 1 ]
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.7 qcsapi_wifi_set_scs_smpl_dwell_time()

```
int qcsapi_wifi_set_scs_smpl_dwell_time (
            const char * ifname,
            uint16_t scs_sample_time )
```

API sets the dwell time for the scs feature ie. the duration in which the busyness of the channel is sampled. Unit is in milliseconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_sample_time	Time during which busyness is sampled.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_smpl_dwell_time <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.8 qcsapi_wifi_set_scs_sample_intv()

```
int qcsapi_wifi_set_scs_sample_intv (
            const char * ifname,
            uint16_t scs_sample_intv )
```

API sets the sample interval for the SCS feature. This duration indicates the duration to wait after which the next off-channel sampling session starts. Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_sample_intv	Time from the previous sample to the next sample.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_smpl_intv <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.9 qcsapi_wifi_set_scs_intf_detect_intv()

```
int qcsapi_wifi_set_scs_intf_detect_intv (
            const char * ifname,
            uint16_t scs_intf_detect_intv )
```

API sets the interference detection interval for the SCS feature. This duration indicates the duration to wait after which the next interference detection session starts. Unit is in seconds.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_intf_detect_intv	Time from the previous interference detection to the next interference detection.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_intf_detect_intv <WiFi interface> <duration>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.10 qcsapi_wifi_set_scs_thrshld()

```
int qcsapi_wifi_set_scs_thrshld (
            const char * ifname,
            const char * scs_param_name,
            uint16_t scs_threshold )
```

API sets the threshold for various parameters that control the SCS feature. Threshold affects the sensitivity of the feature.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_param_name	The threshold by name which is to be set
scs_threshold	The value of the threshold to be set

Returns

0 on success or negative values on error

`call_qcsapi` interface:

`call_qcsapi set_scs_threshold <WiFi interface> <threshold_name> <value>`
threshold name is one of "smpl_pktnum", "smpl_airtime", "intf_low", "intf_high", "intf_ratio", "dfs_margin", "cca_idle", "pmbl_err", "atten_inc", "dfs_reentry", "dfs_reentry_minrate". The unit of "dfs_reentry_minrate" is 100kbps.

Unless an error occurs, the output will be the string `complete`.

9.30.2.11 qcsapi_wifi_set_scs_report_only()

```
int qcsapi_wifi_set_scs_report_only (
            const char * ifname,
            uint16_t scs_report_only )
```

This API controls if SCS should change the channel upon making a decison or just report it.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_report_only	value that indicates if SCS feature should act on thresholds or only report.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

`call_qcsapi set_scs_report_only <WiFi interface> [ 0 | 1 ]`

Unless an error occurs, the output will be the string `complete`.

9.30.2.12 qcsapi_wifi_get_scs_stat_report()

```
int qcsapi_wifi_get_scs_stat_report (
            const char * ifname,
            struct qcsapi_scs_ranking_rpt * scs_rpt )
```

This API reports the evaluation result for all channels for the SCS feature. Statistics like channel, channel metric are returned.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_rpt	return the channel evaluation result.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_report <WiFi interface> all
```

The output will be stats containing channel, channel metric unless an error occurs.

9.30.2.13 qcsapi_wifi_get_scs_score_report()

```
int qcsapi_wifi_get_scs_score_report (
            const char * ifname,
            struct qcsapi_scs_score_rpt * scs_rpt )
```

This API reports the scores of all channels for the SCS feature. Statistics like channel, score are returned.

Note

This API can only be used in AP mode.

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
scs_rpt	return the channel score result.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_report <WiFi interface> score
```

The output will be stats containing channel, score unless an error occurs.

9.30.2.14 qcsapi_wifi_get_scs_currchan_report()

```
int qcsapi_wifi_get_scs_currchan_report (
            const char * ifname,
            struct qcsapi_scs_currchan_rpt * scs_currchan_rpt )
```

This API reports the statistics for the current channel for the SCS feature. Statistics like channel, cca interference are returned.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_currchan_rpt	return the current channel's stats.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_report <WiFi interface> current
```

The output will be stats containing channel, cca interference unless an error occurs.

9.30.2.15 qcsapi_wifi_set_scs_stats()

```
int qcsapi_wifi_set_scs_stats (
            const char * ifname,
            uint16_t start )
```

Start/Stop the SCS stats task.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
start	a value for start/stop indication

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_stats <WiFi interface> [ 0 | 1 ]
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.16 qcsapi_wifi_get_autochan_report()

```
int qcsapi_wifi_get_autochan_report (
            const char * ifname,
            struct qcsapi_autochan_rpt * autochan_rpt )
```

This API reports the initial channel evalution result for Auto Channel feature. Statistics like channel, channel metric are returned.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
autochan_rpt	the initial channel evaluation result.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_report <WiFi interface> autochan
```

The output will be stats containing channel, channel metric unless an error occurs.

9.30.2.17 qcsapi_wifi_set_scs_cca_intf_smth_fctr()

```
int qcsapi_wifi_set_scs_cca_intf_smth_fctr (
            const char * ifname,
            uint8_t smth_fctr_noxp,
            uint8_t smth_fctr_xped )
```

This API controls the degree SCS smoothes sequential cca interference measurement.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
scs_cca_intf_smth_fctr_noxp	value that indicates the smoothing factor for channels once used as working channel.
scs_cca_intf_smth_fctr_xped	value that indicates the smoothing factor for channels never used as working channel.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_cca_intf_smth_fctr <WiFi interface> <smth_fctr_noxp>
<smth_fctr_xped>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.18 qcsapi_wifi_set_scs_chan_mtrc_mrgn()

```
int qcsapi_wifi_set_scs_chan_mtrc_mrgn (
            const char * ifname,
            uint8_t chan_mtrc_mrgn )
```

This API controls the channel metric margin SCS used for channel ranking.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
chan_mtrc_mrgn	value that indicates the channel metric margin.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi set_scs_chan_mtrc_mrgn <WiFi interface> <chan_mtrc_mrgn>
```

Unless an error occurs, the output will be the string `complete`.

9.30.2.19 qcsapi_wifi_get_scs_cca_intf()

```
int qcsapi_wifi_get_scs_cca_intf (
            const char * ifname,
            const qcsapi_unsigned_int the_channel,
            int * p_cca_intf )
```

This API call gets the CCA interference level for a particular channel. The got CCA interference value should be a integer value from -1 to 1000. -1 means no CCA interference data is available, and other values represent CCA interference levels.

Note

This API can only be used on the primary interface (wifi0)

Parameters

ifname	the primary WiFi interface, wifi0 only.
the_channel	the channel for which the CCA interference is returned.
p_cca_intf	return parameter to contain the CCA interference.

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_scs_cca_intf <WiFi interface> <channel>
```

Unless an error occurs, the output will be the CCA interference status.

9.30.2.20 qcsapi_wifi_get_scs_param_report()

```
int qcsapi_wifi_get_scs_param_report (
            const char * ifname,
            struct qcsapi_scs_param_rpt * p_scs_param_rpt,
            uint32_t param_num )
```

This API call gets the configured SCS parameters.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_scs_param_rpt	return parameter to contain the SCS parameters.
param_num	value that indicates parameter numbers needed to be returned

Returns

-EINVAL or other negative values on error, or 0 on success.

`call_qcsapi` interface:

```
call_qcsapi get_scs_param <WiFi interface>
```

Unless an error occurs, the output will include current SCS parameters.

9.30.2.21 qcsapi_wifi_get_scs_dfs_reentry_request()

```
int qcsapi_wifi_get_scs_dfs_reentry_request (
            const char * ifname,
            qcsapi_unsigned_int * p_scs_dfs_reentry_request )
```

Return the current status of the SCS DFS Re-entry request. It could be either 0(not requested) or 1(requested).

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_scs_dfs_reentry_request	value that contains the request level.

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi get_scs_dfs_reentry_request <WiFi interface>
```

The output will be an integer representing the request level unless an error occurs.

9.31 VLAN APIs

These APIs are used to configure, control and obtain status for VLANs within the Quantenna device.

Functions

- int qcsapi_wifi_vlan_config (const char *ifname, qcsapi_vlan_cmd cmd, uint32_t vlanid)

  VLAN configuration for a interface.
- int qcsapi_wifi_show_vlan_config (const char *ifname, struct qcsapi_data_2Kbytes *vcfg, const char *flag)

  get vlan configuration
- int qcsapi_wifi_set_vlan_promisc (int enable)

  Enable and disable VLAN promiscuous mode.

9.31.1 Detailed Description

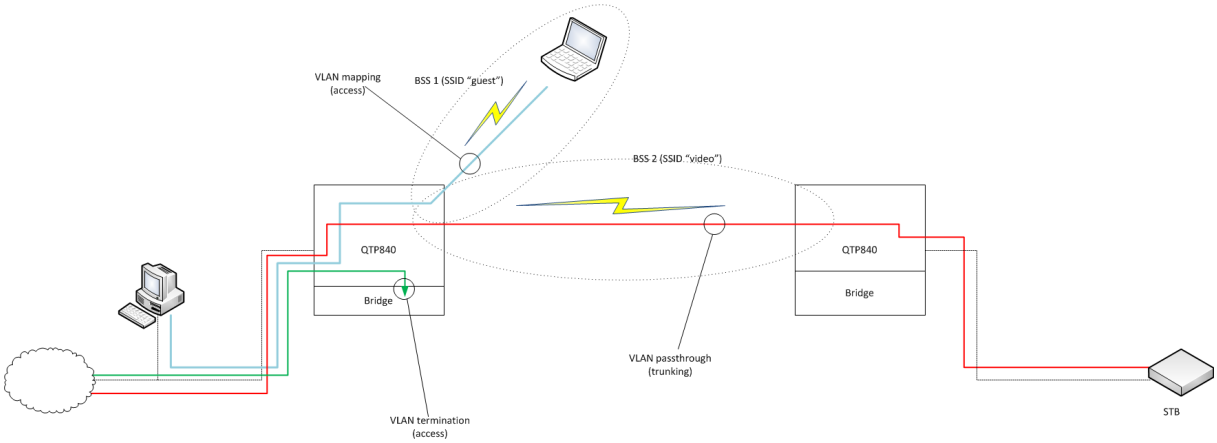
9.31.2 VLAN concepts in wireless networking

This section outlines some key concepts in wireless networking relating specifically to VLANs.

In order to understand the different uses for VLANs, some terminology will be introduced. The following table outlines the three different types of VLAN which apply to wireless systems.

Term	Description
VLAN termination	A type of VLAN which terminates on the Quantenna device Linux host. That is, the Linux host is a member of the given VLAN. In networking terminology, this can be considered the Linux host is part of the VLAN access port.
VLAN passthrough	A type of VLAN which is transparently bridged across 4-address wireless hops (ie, not terminated or mapped). In networking terminology, this can be considered a VLAN trunk port.
VLAN mapping	A type of VLAN which maps to a given BSS, for which the 802.1q header is stripped prior to forwarding the packet to the wireless client. Packets received from the wireless client are tagged with an appropriate 802.1q tag prior to forwarding to the Ethernet interface. In networking terminology, this can be considered a VLAN access port.

The three different modes of VLANs for the Quantenna firmware are shown operating in the following diagram.



9.31.3 Example VLAN configuration - VLAN mapping

This section presents a concrete example of how to map VLANs to configured BSSes (SSIDs). In this case, the Quantenna device being configured is running a pure AP function, mapping clients per BSS into the appropriate VLAN on the Ethernet side of the Quantenna device.

First ensure the Quantenna device is configured with at least two additional BSSes (and corresponding SSIDs). See the section MBSSID APIs for details of configuring multiple BSSes on a single Quantenna device. This example gives a very simple configuration including both BSS creation and VLAN mapping to the SSID.

Second, select two VLAN IDs to map onto the different BSSes. In this example, VLAN 10 and 20 are used.

The script shown in the following code snippet shows how to map VLAN 10 onto SSID Qtn-Open-BSS (with no authentication or encryption), and VLAN 20 onto SSID Qtn-WPA2-BSS (with WPA2 authentication/CCMP encryption enabled).

```
#Enable VLAN functionality
call_qcsapi vlan_config wifi0 enable 0
#Create wifi1, bind to VLAN 10
call_qcsapi wifi_create_bss wifi1
call_qcsapi set_SSID wifi1 "Qtn-Open-BSS"
call_qcsapi set_beacon wifi1 Basic
call_qcsapi vlan_config wifi1 bind 10
#Create wifi2, bind to VLAN 20
call_qcsapi wifi_create_bss wifi2
call_qcsapi set_SSID wifi2 "Qtn-WPA2-BSS"
call_qcsapi set_beacon wifi2 11i
call_qcsapi set_WPA_encryption_modes wifi2 AESEncryption
call_qcsapi set_passphrase wifi2 "This is the passphrase!" 0
call_qcsapi vlan_config wifi2 bind 20
```

9.31.4 Function Documentation

9.31.4.1 qcsapi_wifi_vlan_config()

```
int qcsapi_wifi_vlan_config (
            const char * ifname,
            qcsapi_vlan_cmd cmd,
            uint32_t vlanid )
```

Set vlan configurations on a specific interface

Parameters

ifname	the interface to perform the action on. (e.g. wifi0 or wds0 or eth1_0).
cmd	VLAN command qcsapi_vlan_cmd
VLANID	VLAN identifier, 0 $\sim$ 4095

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi vlan_config <interface> {enable | disable | access | trunk | hybrid
| dynamic | bind | unbind} <VLANID> [tag | untag] [default] [delete]

call_qcsapi vlan_config <interface> enable call_qcsapi vlan_config <interface>
disable  call_qcsapi vlan_config <interface> reset  call_qcsapi vlan_config
<interface> bind 10  call_qcsapi vlan_config <interface> unbind 10  call_↩
qcsapi vlan_config <interface> access 10 call_qcsapi vlan_config <interface>
access 10 delete  call_qcsapi vlan_config <interface> trunk 10  call_qcsapi
vlan_config <interface> trunk 10 default call_qcsapi vlan_config <interface>
trunk 10 delete call_qcsapi vlan_config <interface> trunk 10 default delete
call_qcsapi vlan_config <interface> hybrid 10 call_qcsapi vlan_config <interface>
hybrid 10 default  call_qcsapi vlan_config <interface> hybrid 10 tag  call↩
_qcsapi vlan_config <interface> hybrid 10 untag     call_qcsapi vlan_config
<interface> hybrid 10 delete  call_qcsapi vlan_config <interface> hybrid 10
default delete  call_qcsapi vlan_config <interface> dynamic 1  call_qcsapi
vlan_config <interface> dynamic 0
```

Unless an error occurs, the output will be the string `complete`.

9.31.4.2 qcsapi_wifi_show_vlan_config()

```
int qcsapi_wifi_show_vlan_config (
            const char * ifname,
            struct qcsapi_data_2Kbytes * vcfg,
            const char * flag )
```

This API call is used to retrieve VLAN configuration on a specific interface.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0 or wds0 or eth1_0).
vcfg	the structure to retrieve the VLAN configuration
flag	to imply what kind of configuration we want

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi show_vlan_config
```

Unless an error occurs, the output will be a table of VLAN configuration

9.31.4.3 qcsapi_wifi_set_vlan_promisc()

```
int qcsapi_wifi_set_vlan_promisc (
            int enable )
```

If VLAN promiscuous mode is enabled, all VLAN tagged packets will be sent to the Linux protocol stack.

Note

This API can only be called on an AP device.

Parameters

enabled	set to 1 to enable VLAN promiscuous mode, otherwise set to 0.

Returns

0 if the call succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enable_vlan_promisc <enable>
```

Unless an error occurs, the output will be the string `complete`.

9.32 Statistics APIs

These APIs are used to obtain statistics from the device.

Functions

- int qcsapi_get_interface_stats (const char *ifname, qcsapi_interface_stats *stats)

  Get statistics data of an interface.
- int qcsapi_get_phy_stats (const char *ifname, qcsapi_phy_stats *stats)

  Get latest PHY statistics data of an interface.
- int qcsapi_reset_all_counters (const char *ifname, const uint32_t node_index, int local_remote_flag)

  Reset statistics data of an interface.
- int qcsapi_get_temperature_info (int *temp_exter, int *temp_inter, int *temp_bbic)

  Get RFIC temperature.

9.32.1 Detailed Description

9.32.2 Function Documentation

9.32.2.1 qcsapi_get_interface_stats()

```
int qcsapi_get_interface_stats (
            const char * ifname,
            qcsapi_interface_stats * stats )
```

This API call is used to get interface statistics data.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
stats	return parameter to contain the statistics data of the interface being queried.

Returns

>= 0 on success, < 0 on error. If success, stats contains the statistics data retrieved from the device for the interface.

`call_qcsapi` interface:

```
call_qcsapi get_interface_stats <interface name>
```

Unless an error occurs, the output will be the statistics data of the interface.

9.32.2.2 qcsapi_get_phy_stats()

```
int qcsapi_get_phy_stats (
            const char * ifname,
            qcsapi_phy_stats * stats )
```

This API call is used to get latest PHY statistics data.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
stats	return parameter to contain the statistics data of the interface being queried.

Returns

>= 0 on success, < 0 on error. If success, stats contains the PHY statistics data retrieved from the device for the interface.

`call_qcsapi` interface:

```
call_qcsapi get_phy_stats <interface name>
```

Unless an error occurs, the output will be the PHY statistics data of the interface.

9.32.2.3 qcsapi_reset_all_counters()

```
int qcsapi_reset_all_counters (
            const char * ifname,
            const uint32_t node_index,
            int local_remote_flag )
```

This API call is used to reset interface statistics data.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
node_index	selects the node to operate, it's valid in AP mode when local_remote_flag is set to QCSAPI_REMOTE_NODE.
local_remote_flag	use local flag to reset local all counters, use remote flag to reset all counters on the remote associated STA; set to QCSAPI_LOCAL_NODE or QCSAPI_REMOTE_NODE

Returns

>= 0 on success, < 0 on error. If success,the statistics data of the interface will be cleared.

`call_qcsapi` interface:

```
call_qcsapi reset_all_stats <WiFi interface> <node_index> <QCSAPI_LOCAL_↩
NODE/QCSAPI_REMOTE_NODE>
```

9.32.2.4 qcsapi_get_temperature_info()

```
int qcsapi_get_temperature_info (
             int * temp_exter,
             int * temp_inter,
             int * temp_bbic )
```

Get RFIC temperature

Parameters

temp_exter	Buffer to contain the returned RFIC external temperature.
temp_inter	Buffer to contain the returned RFIC internal temperature.
temp_bbic	Buffer to contain the returned BBIC temperature.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

Note

The return value is a fixed point number, and the actual internal/external temperature value (in degrees Celsius) can be obtained using the following code snippet.

```
float rfic_temperature_extr, rfic_temperature_inter, bbic_temperature;
rfic_temperature_extr = temp_exter / 100000.0f;
rfic_temperature_inter = temp_inter / 1000000.0f;
bbic_temperature = temp_bbic / 1000000.0f;
```

`temp_exter` value returned by this API is guaranteed to be no more than 5 seconds old.

Warning

The values for the RFIC external and internal temperature should not be used for any critical functionality as the values have large part-to-part variance. The values are used for engineering and test purposes only.

`call_qcsapi` interface:

```
call_qcsapi get_temperature
```

9.33 Linux Services APIs

These APIs are used for Linux daemon control.

Functions

- int **qcsapi_telnet_enable** (const qcsapi_unsigned_int onoff)
- int qcsapi_get_service_name_enum (const char *lookup_service, qcsapi_service_name *serv_name)

  Used to find service enum.
- int qcsapi_get_service_action_enum (const char *lookup_action, qcsapi_service_action *serv_action)

  Used to find service action enum.
- int qcsapi_service_control (qcsapi_service_name service, qcsapi_service_action action)

  Start, stop, enable or disable the services.
- int qcsapi_wfa_cert_mode_enable (uint16_t enable)

  Enable and disable features that are not needed for WFA testing.

9.33.1 Detailed Description

9.33.2 Function Documentation

9.33.2.1 qcsapi_get_service_name_enum()

```
int qcsapi_get_service_name_enum (
            const char * lookup_service,
            qcsapi_service_name * serv_name )
```

This is internally used in service_control

9.33.2.2 qcsapi_get_service_action_enum()

```
int qcsapi_get_service_action_enum (
            const char * lookup_action,
            qcsapi_service_action * serv_action )
```

This is internally used in service control

9.33.2.3 qcsapi_service_control()

```
int qcsapi_service_control (
            qcsapi_service_name service,
            qcsapi_service_action action )
```

Turn service on or off.

Parameters

service	
action	start/stop/enable/disable

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi service_control <service_name> <start/stop/enable/disable>
```

Unless an error occurs, the output will be the string `complete`.

9.33.2.4 qcsapi_wfa_cert_mode_enable()

```
int qcsapi_wfa_cert_mode_enable (
            uint16_t enable )
```

Turn WFA certification mode on or off.

Parameters

enable	a value turn the WFA certification mode on/off

Returns

0 on success or negative values on error

`call_qcsapi` interface:

```
call_qcsapi wfa_cert <1|0>
```

Unless an error occurs, the output will be the string `complete`.

9.34 Quantenna Traffic Management (QTM) APIs

These APIs are used for Quantenna Traffic Management (QTM) configuration.

Functions

- int qcsapi_qtm_get_state (const char *ifname, unsigned int param, unsigned int *value)

  Get QTM status.
- int qcsapi_qtm_get_state_all (const char *ifname, struct qcsapi_data_128bytes *value, unsigned int max)

  Get all QTM status.
- int qcsapi_qtm_set_state (const char *ifname, unsigned int param, unsigned int value)

  Set QTM status.
- int qcsapi_qtm_get_config (const char *ifname, unsigned int param, unsigned int *value)

  Get QTM config option.
- int qcsapi_qtm_get_config_all (const char *ifname, struct qcsapi_data_1Kbytes *value, unsigned int max)

  Get all QTM config options.
- int qcsapi_qtm_set_config (const char *ifname, unsigned int param, unsigned int value)

  Set QTM config option.
- int qcsapi_qtm_add_rule (const char *ifname, const struct qcsapi_data_128bytes *entry)

  Add QTM rule.
- int qcsapi_qtm_del_rule (const char *ifname, const struct qcsapi_data_128bytes *entry)

  Delete QTM rule.
- int qcsapi_qtm_del_rule_index (const char *ifname, unsigned int index)

  Delete a QTM rule by index.
- int qcsapi_qtm_get_rule (const char *ifname, struct qcsapi_data_3Kbytes *entries, unsigned int max_entries)

  Read QTM rules.
- int qcsapi_qtm_get_strm (const char *ifname, struct qcsapi_data_4Kbytes *strms, unsigned int max_entries, int show_all)

  Read QTM streams.
- int qcsapi_qtm_get_stats (const char *ifname, struct qcsapi_data_512bytes *stats)

  Read QTM statistics.
- int qcsapi_qtm_get_inactive_flags (const char *ifname, unsigned long *flags)

  Read QTM inactive flags.
- int qcsapi_qtm_safe_get_state_all (const char *ifname, struct qcsapi_int_array32 *value, unsigned int max)

  Get all QTM status.
- int qcsapi_qtm_safe_get_config_all (const char *ifname, struct qcsapi_int_array256 *value, unsigned int max)

  Get all QTM config options.
- int qcsapi_qtm_safe_add_rule (const char *ifname, const struct qcsapi_int_array32 *entry)

  Add QTM rule.
- int qcsapi_qtm_safe_del_rule (const char *ifname, const struct qcsapi_int_array32 *entry)

  Delete QTM rule.
- int qcsapi_qtm_safe_get_rule (const char *ifname, struct qcsapi_int_array768 *entries, unsigned int max_↩ entries)

  Read QTM rules.
- int qcsapi_qtm_safe_get_strm (const char *ifname, struct qvsp_strms *strms, int show_all)

  Read QTM streams.
- int qcsapi_qtm_safe_get_stats (const char *ifname, struct qcsapi_int_array128 *stats)

  Read QTM statistics.

9.34.1 Detailed Description

9.34.2 Function Documentation

9.34.2.1 qcsapi_qtm_get_state()

```
int qcsapi_qtm_get_state (
            const char * ifname,
            unsigned int param,
            unsigned int * value )
```

This API obtains QTM runtime status and flags

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param	item to query status of. Valid values are: • QVSP_STATE_ENABLE - query whether qtm is enabled • QVSP_STATE_FAT - query most recently reported free air time
value	result memory

Returns

0 on success, negative on error.

9.34.2.2 qcsapi_qtm_get_state_all()

```
int qcsapi_qtm_get_state_all (
            const char * ifname,
            struct qcsapi_data_128bytes * value,
            unsigned int max )
```

This API obtains all QTM runtime status and flags with 1 invocation

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	pointer to an array of unsigned ints for value storage
max	maximum number of unsigned ints to return, typically QVSP_STATE_READ_MAX

Returns

0 on success, negative on error.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_get_state_all.

9.34.2.3 qcsapi_qtm_set_state()

```
int qcsapi_qtm_set_state (
            const char * ifname,
            unsigned int param,
            unsigned int value )
```

This API handles QTM enable/disable, test settings and reset

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param	item to query status of. Valid values are: • QVSP_STATE_ENABLE - enable (value nonzero) / disable (value zero) QTM • QVSP_STATE_RESET - reset VSP stream accounting (value unused) • QVSP_STATE_TEST_FAT - set a FAT value in order to simulate undersubscription or oversubscription (value is FAT to set)
value	context dependant parameter value

Parts of this API can be called via `call_qcsapi`.

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> enable` `call_qcsapi qtm <ifname> disable` `call_qcsapi qtm <ifname> reset` `call_qcsapi qtm <ifname> test fat <value>`

9.34.2.4 qcsapi_qtm_get_config()

```
int qcsapi_qtm_get_config (
            const char * ifname,
            unsigned int param,
            unsigned int * value )
```

This API obtains QTM configuration options

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param	configuration option to query
value	result memory

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> get <param>
```

This will print the parameter obtained if successful, or an error message on failure.

9.34.2.5 qcsapi_qtm_get_config_all()

```
int qcsapi_qtm_get_config_all (
            const char * ifname,
            struct qcsapi_data_1Kbytes * value,
            unsigned int max )
```

This API stores all QTM configuration options into an array

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	pointer to an array of unsigned ints for value storage
max	maximum number of unsigned ints to return, typically QVSP_STATE_READ_MAX

Returns

0 on success, negative on error.

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show config` will call this API, and some others.

Unless an error occurs, the output will be all configuration options, all whitelist entries and all rules.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_get_config_all.

9.34.2.6 qcsapi_qtm_set_config()

```
int qcsapi_qtm_set_config (
            const char * ifname,
            unsigned int param,
            unsigned int value )
```

This API sets QTM configuration options

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
param	configuration option to modify
value	value to set

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> set <param> <value>
```

Output is silent with a return code of zero if successful, and an error message on failure.

9.34.2.7 qcsapi_qtm_add_rule()

```
int qcsapi_qtm_add_rule (
            const char * ifname,
            const struct qcsapi_data_128bytes * entry )
```

Add a QTM rule. Rules determine which streams to drop in an oversubscription event.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entry	rule to add

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> rule add <p1> <v1> [<pn> <vn>]...
```

Where <p1> <v1> are rule parameter name/value pairs. Many pairs can be used in one rule.

Output is silent with a return code of zero if successful, and an error message on failure.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_add_rule.

9.34.2.8 qcsapi_qtm_del_rule()

```
int qcsapi_qtm_del_rule (
            const char * ifname,
            const struct qcsapi_data_128bytes * entry )
```

Delete a QTM rule.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entry	rule to delete

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> rule add <p1> <v1> [<pn> <vn>]...
```

Where <p1> <v1> are rule parameter name/value pairs. Many pairs can be used in one rule.

Output is silent with a return code of zero if successful, and an error message on failure.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_del_rule.

9.34.2.9 qcsapi_qtm_del_rule_index()

```
int qcsapi_qtm_del_rule_index (
            const char * ifname,
            unsigned int index )
```

Delete a QTM rule by index

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
index	index of the entry to delete, starting from 1

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> rule del <index>
```

```
Output is silent with a return code of zero if successful, and an error
message on failure.
```

9.34.2.10 qcsapi_qtm_get_rule()

```
int qcsapi_qtm_get_rule (
            const char * ifname,
            struct qcsapi_data_3Kbytes * entries,
            unsigned int max_entries )
```

Read QTM rules

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entries	an array of rule structures for result storage
max_entries	the length of the array passed in 'entries'

Returns

Number of rules currently configured on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show config` will call this API, and some others.

Unless an error occurs, the output will be all configuration options, all whitelist entries and all rules.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_get_rule.

9.34.2.11 qcsapi_qtm_get_strm()

```
int qcsapi_qtm_get_strm (
            const char * ifname,
            struct qcsapi_data_4Kbytes * strms,
            unsigned int max_entries,
            int show_all )
```

Read QTM streams

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entries	an buffer of stream structures for result storage
max_entries	the length of the array passed in 'entries'
show_all	1 to return all streams, or 0 to return only enabled, pre-enabled and disabled streams

Returns

Number of streams currently configured on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show [all]` will call this API, and some others.

Unless an error occurs, the output will be status, and current high throughput streams. Usage of the argument 'all' adds low throughput streams.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_get_strm.

9.34.2.12 qcsapi_qtm_get_stats()

```
int qcsapi_qtm_get_stats (
            const char * ifname,
            struct qcsapi_data_512bytes * stats )
```

Read QTM statistics

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
stats	result memory

Returns

0 on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show stats`

The output will show various statistics on success, and an error message on failure.

Note

This API is deprecated and replaced with qcsapi_qtm_safe_get_stats.

9.34.2.13 qcsapi_qtm_get_inactive_flags()

```
int qcsapi_qtm_get_inactive_flags (
            const char * ifname,
            unsigned long * flags )
```

Read QTM inactive flags

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
flags	result memory

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> show
```

The output will show the inactive reason if QTM is inactive.

9.34.2.14 qcsapi_qtm_safe_get_state_all()

```
int qcsapi_qtm_safe_get_state_all (
            const char * ifname,
            struct qcsapi_int_array32 * value,
            unsigned int max )
```

This API obtains all QTM runtime status and flags with 1 invocation

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	pointer to an integer array for value storage
max	maximum number of unsigned ints to return, typically QVSP_STATE_READ_MAX

Returns

0 on success, negative on error.

9.34.2.15 qcsapi_qtm_safe_get_config_all()

```
int qcsapi_qtm_safe_get_config_all (
            const char * ifname,
            struct qcsapi_int_array256 * value,
            unsigned int max )
```

This API stores all QTM configuration options into an array

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
value	pointer to an integer array for value storage
max	maximum number of unsigned ints to return, typically QVSP_STATE_READ_MAX

Returns

0 on success, negative on error.

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show config` will call this API, and some others.

Unless an error occurs, the output will be all configuration options, all whitelist entries and all rules.

9.34.2.16 qcsapi_qtm_safe_add_rule()

```
int qcsapi_qtm_safe_add_rule (
            const char * ifname,
            const struct qcsapi_int_array32 * entry )
```

Add a QTM rule. Rules determine which streams to drop in an oversubscription event.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entry	rule to add

Returns

0 on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> rule add <p1> <v1> [<pn> <vn>]...`

Where <p1> <v1> are rule parameter name/value pairs. Many pairs can be used in one rule.

Output is silent with a return code of zero if successful, and an error message on failure.

9.34.2.17 qcsapi_qtm_safe_del_rule()

```
int qcsapi_qtm_safe_del_rule (
            const char * ifname,
            const struct qcsapi_int_array32 * entry )
```

Delete a QTM rule.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entry	rule to delete

Returns

0 on success, negative on error

`call_qcsapi` interface:

```
call_qcsapi qtm <ifname> rule del <p1> <v1> [<pn> <vn>]...
```

Where <p1> <v1> are rule parameter name/value pairs. Many pairs can be used in one rule.

Output is silent with a return code of zero if successful, and an error message on failure.

9.34.2.18 qcsapi_qtm_safe_get_rule()

```
int qcsapi_qtm_safe_get_rule (
            const char * ifname,
            struct qcsapi_int_array768 * entries,
            unsigned int max_entries )
```

Read QTM rules

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
entries	an array of rule structures for result storage
max_entries	the length of the array passed in 'entries'

Returns

Number of rules currently configured on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show config` will call this API, and some others.

Unless an error occurs, the output will be all configuration options, all whitelist entries and all rules.

9.34.2.19 qcsapi_qtm_safe_get_strm()

```
int qcsapi_qtm_safe_get_strm (
            const char * ifname,
            struct qvsp_strms * strms,
            int show_all )
```

Read QTM streams

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
strms	buffer to receive stream data
show_all	1 to return all streams, or 0 to return only enabled, pre-enabled and disabled streams

Returns

Number of streams currently configured on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show [all]` will call this API, and some others.

Unless an error occurs, the output will be status, and current high throughput streams. Usage of the argument 'all' adds low throughput streams.

Note

This API should be used with care because the return buffer consumes ~21KB of memory, which could lead to memory exhaustion.

9.34.2.20 qcsapi_qtm_safe_get_stats()

```
int qcsapi_qtm_safe_get_stats (
            const char * ifname,
            struct qcsapi_int_array128 * stats )
```

Read QTM statistics

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
stats	result memory

Returns

0 on success, negative on error

`call_qcsapi` interface:

`call_qcsapi qtm <ifname> show stats`

The output will show various statistics on success, and an error message on failure.

9.35 TDLS APIs

These APIs are used for configuration and control of TDLS.

Functions

- int qcsapi_wifi_enable_tdls (const char *ifname, uint32_t enable_tdls)

  enable TDLS
- int qcsapi_wifi_enable_tdls_over_qhop (const char *ifname, uint32_t tdls_over_qhop_en)

  enable TDLS over Qhop
- int qcsapi_wifi_get_tdls_status (const char *ifname, uint32_t *p_tdls_status)

  get TDLS status
- int qcsapi_wifi_set_tdls_params (const char *ifname, qcsapi_tdls_type type, int param_value)

  set TDLS parameters
- int qcsapi_wifi_get_tdls_params (const char *ifname, qcsapi_tdls_type type, int *p_value)

  get TDLS parameters
- int qcsapi_wifi_tdls_operate (const char *ifname, qcsapi_tdls_oper operate, const char *mac_addr_str, int cs_interval)

  excute TDLS operation

9.35.1 Detailed Description

9.35.2 Function Documentation

9.35.2.1 qcsapi_wifi_enable_tdls()

```
int qcsapi_wifi_enable_tdls (
            const char * ifname,
            uint32_t enable_tdls )
```

This API call is used to enable or disable tdls

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
enable_tdls	disable or enable tdls, 0 disable, 1 enable

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enable_tdls <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.35.2.2 qcsapi_wifi_enable_tdls_over_qhop()

```
int qcsapi_wifi_enable_tdls_over_qhop (
            const char * ifname,
            uint32_t tdls_over_qhop_en )
```

This API call is used to enable or disable tdls over Qhop

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
tdls_over_qhop_en	disable or enable tdls over qhop, 0 disable, 1 enable

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi enable_tdls_over_qhop <WiFi interface> <0 | 1>
```

Unless an error occurs, the output will be the string `complete`.

9.35.2.3 qcsapi_wifi_get_tdls_status()

```
int qcsapi_wifi_get_tdls_status (
            const char * ifname,
            uint32_t * p_tdls_status )
```

This API call is used to retrieve TDLS status, if TDLS is enable, also display current TDLS path select mode

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_tdls_status	return parameter to store the TDLS status

Returns

>= 0 on success, < 0 on error. If success, TDLS status contain

`call_qcsapi` interface:

```
call_qcsapi get_tdls_status <WiFi interface>
```

Unless an error occurs, the output will be the TDLS related status.

9.35.2.4 qcsapi_wifi_set_tdls_params()

```
int qcsapi_wifi_set_tdls_params (
            const char * ifname,
            qcsapi_tdls_type type,
            int param_value )
```

This API call is used to set TDLS parameters

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	tdls parameter type, used to indentify tdls parameter
param_value	parameter value set to system

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_tdls_params <WiFi interface> <parameter name> <value>
```

where `parameter name` is one of `link_timeout_time`, `link_weight`, `disc_interval`, `training_pkt_cnt`, `path_sel_pps_thrshld`, `path_sel_rate_thrshld` `min_valid_rssi` `link_switch_ints` `phy_rate_weight` `mode` `indication_window` `node_life_cycle` `chan_switch_mode` `chan_switch_off_chan` or `chan_switch_off_chan_bw`

Unless an error occurs, the output will be the string `complete`.

9.35.2.5 qcsapi_wifi_get_tdls_params()

```
int qcsapi_wifi_get_tdls_params (
            const char * ifname,
            qcsapi_tdls_type type,
            int * p_value )
```

This API call is used to retrieve TDLS parameters

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
type	tdls parameter type, used to indentify tdls parameter
p_value	return parameter to store the TDLS parameter value

Returns

>= 0 on success, < 0 on error. If success, TDLS parameter value contain

`call_qcsapi` interface:

```
call_qcsapi get_tdls_params <WiFi interface>
```

Unless an error occurs, the output will be the TDLS related parameter value.

9.35.2.6 qcsapi_wifi_tdls_operate()

```
int qcsapi_wifi_tdls_operate (
            const char * ifname,
            qcsapi_tdls_oper operate,
            const char * mac_addr_str,
            int cs_interval )
```

This API call is used to excute TDLS operation

\Note: This API is only used on a station.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
operate	TDLS operation type, indentify TDLS operation
mac_addr_str	peer station mac address string, its format is "xx:xx:xx:xx:xx:xx"
cs_interval	channel switch interval as milliseconds, only required by operation switch_chan, 0 indicates to stop the channel switch.

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi tdls_operate <WiFi interface> <operation name> <mac address>
[cs_interval]
```

where `operation name` is one of `discover`, `setup`, `teardown` or `switch_chan`

Unless an error occurs, the output will be the string `complete`.

9.36 Multi-user MIMO APIs

These APIs are used for debugging and display of MU details.

Functions

- int qcsapi_wifi_set_enable_mu (const char *ifname, const unsigned int mu_enable)

  Enable/disable MU-MIMO functionality on AP.
- int qcsapi_wifi_get_enable_mu (const char *ifname, unsigned int *mu_enable)

  Get the status of MU-MIMO, if it is enabled or not.
- int qcsapi_wifi_set_mu_use_precode (const char *ifname, const unsigned int grp, const unsigned int prec_enable)

  Enable/disable MU-MIMO precoding matrix for the group on AP.
- int qcsapi_wifi_get_mu_use_precode (const char *ifname, const unsigned int grp, unsigned int *prec_enable)

  Get the status of MU-MIMO precoding matric for the group, if it is enabled or not.
- int qcsapi_wifi_set_mu_use_eq (const char *ifname, const unsigned int eq_enable)

  Enable/disable MU equalizer on STA.
- int qcsapi_wifi_get_mu_use_eq (const char *ifname, unsigned int *meq_enable)

  Get the status of MU equalizer, if it is enabled or not.
- int qcsapi_wifi_get_mu_groups (const char *ifname, char *buf, const unsigned int size)

  Get information about MU-MIMO groups formed.

9.36.1 Detailed Description

9.36.2 Function Documentation

9.36.2.1 qcsapi_wifi_set_enable_mu()

```
int qcsapi_wifi_set_enable_mu (
            const char * ifname,
            const unsigned int mu_enable )
```

This API call is used to enable/disable MU-MIMO functionality on AP

Note

This API only applies for an AP.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mu_enable	1 to enable MU-MIMO, 0 to disable it

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_enable_mu <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.36.2.2 qcsapi_wifi_get_enable_mu()

```
int qcsapi_wifi_get_enable_mu (
            const char * ifname,
            unsigned int * mu_enable )
```

This API call is used to get the the status of MU-MIMO

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mu_enable	return value storing a flag showing if MU-MIMO is enabled or not

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_enable_mu <WiFi interface>
```

Unless an error occurs, the output will be the MU-MIMO enable flag

9.36.2.3 qcsapi_wifi_set_mu_use_precode()

```
int qcsapi_wifi_set_mu_use_precode (
            const char * ifname,
            const unsigned int grp,
            const unsigned int prec_enable )
```

This API call is used to enable/disable MU-MIMO precoding matrix for the group

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
grp	MU-MIMO group ID, the valid range is [1-62]
prec_enable	1 to enable MU-MIMO precoding matrix for the group, 0 to disable it

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_mu_use_precode <WiFi interface> <group> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.36.2.4 qcsapi_wifi_get_mu_use_precode()

```
int qcsapi_wifi_get_mu_use_precode (
            const char * ifname,
            const unsigned int grp,
            unsigned int * prec_enable )
```

This API call is used to get the the status of MU-MIMO precoding matrix

Note

This API only applies for an AP.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
grp	MU-MIMO group ID, the valid range is [1-62]
prec_enable	return value storing a flag showing if MU-MIMO precoding matrix is enabled or not

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_mu_use_precode <WiFi interface> <group>
```

Unless an error occurs, the output will be the MU-MIMO enable flag

9.36.2.5 qcsapi_wifi_set_mu_use_eq()

```
int qcsapi_wifi_set_mu_use_eq (
            const char * ifname,
            const unsigned int eq_enable )
```

This API call is used to enable/disable MU-MIMO equalizer on STA

Note

This API only applies for an STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
eq_enable	1 to enable equalizer, 0 to disable it

Returns

0 if the command succeeded.

A negative value if an error occurred. See QCSAPI Return Values for error codes and messages.

`call_qcsapi` interface:

```
call_qcsapi set_mu_use_eq <WiFi interface> <value>
```

Unless an error occurs, the output will be the string `complete`.

9.36.2.6 qcsapi_wifi_get_mu_use_eq()

```
int qcsapi_wifi_get_mu_use_eq (
            const char * ifname,
            unsigned int * meq_enable )
```

This API call is used to get the the status of MU equalizer

Note

This API only applies for an STA.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
mu_enable	return value storing a flag showing if MU equalizer is enabled or not

Returns

>= 0 on success, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_mu_use_eq <WiFi interface>
```

Unless an error occurs, the output will be the MU equalizer enable flag

9.36.2.7 qcsapi_wifi_get_mu_groups()

```
int qcsapi_wifi_get_mu_groups (
            const char * ifname,
            char * buf,
            const unsigned int size )
```

This API call is used to get information about MU-MIMO groups

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
buf	pointer to a buffer where the resulted information is placed to
size	size of the buffer

Returns

>= 0 the number of bytes returned in the buf, < 0 on error.

`call_qcsapi` interface:

```
call_qcsapi get_mu_groups <WiFi interface>
```

Unless an error occurs, the output on AP will be MU groups information in the following form GRP ID: 1 update cnt 304 Enabled Rank: 44468 AID0: 0x0001 AID1: 0x0004 IDX0: 5 IDX1: 8 u0_1ss_u1_1ss: 0x0 u0_2ss_u1_1ss: 0x21 u0_3ss_u1_1ss: 0x52 u0_1ss_u1_2ss: 0x93 u0_1ss_u1_3ss: 0xc4 u0_2ss_u1_2ss: 0x105 .* The same table is repeated for each existing groups group For the STA the output will be AP GRP ID: 1 update cnt 0 User pos = 0 with AID = 0x0004 The same table is reapeated for every group the STA belongs to

9.37 Wake on WLAN (WoWLAN) APIs

These APIs are used for Wake on WLAN configuration and control.

Functions

- int qcsapi_wowlan_set_match_type (const char *ifname, const uint32_t wowlan_match)

  set WOWLAN match type
- int qcsapi_wowlan_set_L2_type (const char *ifname, const uint32_t ether_type)

  set WOWLAN L2 ether type
- int qcsapi_wowlan_set_udp_port (const char *ifname, const uint32_t udp_port)

  set WOWLAN L3 UDP destination port
- int qcsapi_wowlan_set_magic_pattern (const char *ifname, struct qcsapi_data_256bytes *pattern, uint32_t len)

  set user self-defined WOWLAN match pattern
- int qcsapi_wifi_wowlan_get_host_state (const char *ifname, uint16_t *p_value, uint32_t *len)

  Get host CPU's power save state.
- int qcsapi_wifi_wowlan_get_match_type (const char *ifname, uint16_t *p_value, uint32_t *len)

  Get WOWLAN match type. This API is used to get which match type is used for current WOWLAN filtering.
- int qcsapi_wifi_wowlan_get_l2_type (const char *ifname, uint16_t *p_value, uint32_t *len)

  Get WOWLAN ether type value.
- int qcsapi_wifi_wowlan_get_udp_port (const char *ifname, uint16_t *p_value, uint32_t *len)

  Get WOWLAN UDP destination port.
- int qcsapi_wifi_wowlan_get_magic_pattern (const char *ifname, struct qcsapi_data_256bytes *p_value, uint32_t *len)

  Get WOWLAN magci pattern.

9.37.1 Detailed Description

9.37.2 Function Documentation

9.37.2.1 qcsapi_wowlan_set_match_type()

```
int qcsapi_wowlan_set_match_type (
            const char * ifname,
            const uint32_t wowlan_match )
```

This API call is used to set which type to use to match WOWLAN packet

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
wowlan_match	'1' (L2 ether type) or '2' (L3 UDP), default is 0 match either of them

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi wowlan_match_type <WiFi interface> {0 | 1 | 2}
```

9.37.2.2 qcsapi_wowlan_set_L2_type()

```
int qcsapi_wowlan_set_L2_type (
            const char * ifname,
            const uint32_t ether_type )
```

This API call is used to set the ehter type value when using L2 to do matching

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
ether_type	ether type value. 0x0842 will be used by default.

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi wowlan_L2_type <WiFi interface> <\ether type>
```

9.37.2.3 qcsapi_wowlan_set_udp_port()

```
int qcsapi_wowlan_set_udp_port (
            const char * ifname,
            const uint32_t udp_port )
```

This API call is used to set UDP destination port when using UDP to do matching

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
udp_port	UDP destination port value. By default 7 or 9 will be used

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi wowlan_udp_port <WiFi interface> <\udp dest port>
```

9.37.2.4 qcsapi_wowlan_set_magic_pattern()

```
int qcsapi_wowlan_set_magic_pattern (
            const char * ifname,
            struct qcsapi_data_256bytes * pattern,
            uint32_t len )
```

This API call is used to set user self-defined pattern.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
pattern	pattern array, 256 bytes in total length. Default format is 6 bytes of all 255 (FF FF FF FF FF FF in hexadecimal), followed by sixteen repetitions of BSSID or host CPU's MAC address
len	length of pattern

Note

This API works across all WiFi interfaces.

Returns

>= 0 on success, < 0 on error.

\call_qcsapi

```
call_qcsapi wowlan_pattern <WiFi interface> <pattern> <len>
```

9.37.2.5 qcsapi_wifi_wowlan_get_host_state()

```
int qcsapi_wifi_wowlan_get_host_state (
            const char * ifname,
            uint16_t * p_value,
            uint32_t * len )
```

This API is used to get host CPU's power save state.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_value	Buffer contains state of host CPU, 1: host in power save state, 0 : not.
len	Buffer contains the length of the return parameter value.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi wowlan_get_host_state <WiFi interface>
```

Unless an error occurs, the output will be the value of the host state.

9.37.2.6 qcsapi_wifi_wowlan_get_match_type()

```
int qcsapi_wifi_wowlan_get_match_type (
            const char * ifname,
            uint16_t * p_value,
            uint32_t * len )
```

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_value	Buffer contains match type value, 0: default, 1: L2 ether type matching, 2: UDP port matching.
len	Buffer contains the length of the return parameter value.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi wowlan_get_match_type <WiFi interface>
```

Unless an error occurs, the output will be the value of the match type.

9.37.2.7 qcsapi_wifi_wowlan_get_l2_type()

```
int qcsapi_wifi_wowlan_get_l2_type (
            const char * ifname,
            uint16_t * p_value,
            uint32_t * len )
```

This API is used to ether type value used for current WOWLAN filtering.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_value	Buffer contains ether type value.
len	Buffer contains the length of the return parameter value.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi wowlan_get_L2_type <WiFi interface>
```

Unless an error occurs, the output will be the value of the ether type.

9.37.2.8 qcsapi_wifi_wowlan_get_udp_port()

```
int qcsapi_wifi_wowlan_get_udp_port (
            const char * ifname,
            uint16_t * p_value,
            uint32_t * len )
```

This API is used to get UDP destination port value used for current WOWLAN filtering.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_value	Buffer contains udp port value.
len	Buffer contains the length of the return parameter value.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi wowlan_get_udp_port <WiFi interface>
```

Unless an error occurs, the output will be the value of the udp port.

9.37.2.9 qcsapi_wifi_wowlan_get_magic_pattern()

```
int qcsapi_wifi_wowlan_get_magic_pattern (
            const char * ifname,
            struct qcsapi_data_256bytes * p_value,
            uint32_t * len )
```

This API is used to get magic pattern used for current WOWLAN filtering.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
p_value	Buffer contains magic pattern in the format of "010203998877".
len	Buffer contains the length of the return parameter value.

Returns

negative value on error, 0 on success.

`call_qcsapi` interface:

```
call_qcsapi wowlan_get_pattern <WiFi interface>
```

Unless an error occurs, the output will be the value of the magic pattern and its length.

9.38 Android APIs

These APIs are used for integrating QTN firmware with Android host.

Functions

- int qcsapi_wifi_get_scan_IEs (const char *ifname, struct qcsapi_data_1Kbytes *buf, uint32_t block_id, uint32_t *ulength)

  this API is used to get scan IEs from the scan result.

9.38.1 Detailed Description

9.38.2 Function Documentation

9.38.2.1 qcsapi_wifi_get_scan_IEs()

```
int qcsapi_wifi_get_scan_IEs (
            const char * ifname,
            struct qcsapi_data_1Kbytes * buf,
            uint32_t block_id,
            uint32_t * ulength )
```

Currently this API is only used for Android support, user should not call this API explicitly. And there is no call_↩ qcsapi interface is provided for this API, only C interface is available. Android wireless driver need the scan result IE for further process. This API will save and send IE buf. And The IE buf will be zipped before sending to host for processing. Please consult with Quantenna for the usage of the API.

Parameters

ifname	the interface to perform the action on. (e.g. wifi0).
buf	buffer pointer for tranferring the IE.
block↩ *_id*	the block ID in the IE buf, each block is 1K Bytes.
ulength	the IE buffer size readed from the IE, it should be less than 1K for the last block.

Returns

>=0 on success, <0 on error.

Unless an error occurs, the buf will contains the IE buf block.

9.39 QWEAPIs

Functions

- int qcsapi_qwe_command (const char ∗command, const char ∗param1, const char ∗param2, const char ∗param3, char ∗output, const unsigned int max_len)

  perform QWE commands.

9.39.1 Detailed Description

9.39.2 Function Documentation

9.39.2.1 qcsapi_qwe_command()

```
int qcsapi_qwe_command (
            const char * command,
            const char * param1,
            const char * param2,
            const char * param3,
            char * output,
            const unsigned int max_len )
```

QWE is the new Quantenna Wireless Extension Platform Abstract Layer which is introduced to control 3rd party 2.4G. The purpose of QWE is to provide unified interface of 2.4G interface to upper layer such as Web GUI. QWE provides two different commands "qweconfig" and "qweaction", "qweconfig" will get/set the 2.4G configurations without take effects immediately. "qweaction" will act some operations immediately such as take effect of 2.4G configuration, WPS PBC and get all kinds of status.

This API make QWE to be compatible with QCSAPI, by this API 3rd party 2.4G can be controlled by QCSAPI RPC remotely in case QV860 is designed as a module instead of video bridge.

Parameters

command	The QWE command to perform, it can be "qweconfig" or "qweaction" or "help".
param1	The parameter 1 for the QWE command, the meaning of this parameter depends on command.
param2	The parameter 2 for the QWE command, the meaning of this parameter depends on command.
param3	The parameter 3 for the QWE command, the meaning of this parameter depends on command.
output	Address of the buffer to receive the output of the command, it's recommended that the size of buffer is 1024 bytes.
max_len	Maximum number of characters that can be written to the parameter `output`

Returns

= 0 on success, = 1 on failure with error message in output, < 0 on failure without error message.

`call_qcsapi` interface:

```
call_qcsapi qwe qweconfig <command> [<param>] [<value>]    call_qcsapi qwe
qweaction <device> <command> [<argument>]
```

You can use command "call_qcsapi qwe help" to get the usage.

Chapter 10

Data Structure Documentation

10.1 _qcsapi_calcmd_rssi_rsp Struct Reference

Data Fields

- int32_t **value** [QCSAPI_QDRV_NUM_RF_STREAMS]

10.1.1 Detailed Description

Retrieve values of rssi on all antennas for calcmd

10.2 _qcsapi_calcmd_tx_power_rsp Struct Reference

Data Fields

- uint32_t **value** [QCSAPI_QDRV_NUM_RF_STREAMS]

10.2.1 Detailed Description

Retrieve values of tx_power on all antennas for calcmd

10.3 _qcsapi_chan_disabled_data Struct Reference

Data Fields

- uint8_t **chan** [QCSAPI_MAX_CHANNEL]
- uint32_t **list_len**
- uint8_t **flag**
- uint8_t **dir**

10.4 _qcsapi_csw_record Struct Reference

Data Fields

- uint32_t cnt
- int32_t index
- uint32_t channel [QCSAPI_CSW_MAX_RECORDS]
- uint32_t timestamp [QCSAPI_CSW_MAX_RECORDS]
- uint32_t reason [QCSAPI_CSW_MAX_RECORDS]
- qcsapi_mac_addr csw_record_mac [QCSAPI_CSW_MAX_RECORDS]

10.4.1 Detailed Description

Channel switch history record

10.4.2 Field Documentation

10.4.2.1 cnt

```
uint32_t cnt
```

Entry number. Maximum value is QCSAPI_CSW_MAX_RECORDS.

10.4.2.2 index

```
int32_t index
```

Index of the latest channel change.

10.4.2.3 channel

```
uint32_t channel[QCSAPI_CSW_MAX_RECORDS]
```

Channel number which the device switch to. If the value is 0, it means the record is invalid.

10.4.2.4 timestamp

```
uint32_t timestamp[QCSAPI_CSW_MAX_RECORDS]
```

Time when the channel change happens.

10.4.2.5 reason

```
uint32_t reason[QCSAPI_CSW_MAX_RECORDS]
```

Reason for channel change. Possible values are enumerated by ieee80211_csw_reason

10.4.2.6 csw_record_mac

```
qcsapi_mac_addr csw_record_mac[QCSAPI_CSW_MAX_RECORDS]
```

The MAC address of the associated station which required the AP to change channel via SCS mechanism.

10.5 _qcsapi_disconn_info Struct Reference

Data Fields

- uint32_t asso_sta_count
- uint32_t disconn_count
- uint32_t sequence
- uint32_t up_time
- uint32_t resetflag

10.5.1 Detailed Description

Connection and Disconnecttion count information

10.5.2 Field Documentation

10.5.2.1 asso_sta_count

```
uint32_t asso_sta_count
```

This indicates number of stations connect to this device.

10.5.2.2 disconn_count

```
uint32_t disconn_count
```

Count of disconnect event.

10.5.2.3 sequence

```
uint32_t sequence
```

Sequence to query disconnect count.

10.5.2.4 up_time

```
uint32_t up_time
```

Time elapses since device boot up.

10.5.2.5 resetflag

```
uint32_t resetflag
```

If resetflag is set to TRUE, member disconn_count and sequence will be set to 0.

10.6 _qcsapi_dscp2ac_data Struct Reference

Data Fields

- uint8_t ip_dscp_list [64]
- uint8_t list_len
- uint8_t ac

10.6.1 Detailed Description

Data should be set to the dscp to ac mapping

10.6.2 Field Documentation

10.6.2.1 ip_dscp_list

```
uint8_t ip_dscp_list[64]
```

dscp value to be mapped

10.6.2.2 list_len

```
uint8_t list_len
```

Length of DSCP list

10.6.2.3 ac

```
uint8_t ac
```

WME Access Class

10.7 _qcsapi_interface_stats Struct Reference

Structure to contain per interface statistics.

Data Fields

- uint64_t tx_bytes
- uint32_t tx_pkts
- uint32_t tx_discard
- uint32_t tx_wifi_drop [WMM_AC_NUM]
- uint32_t tx_wifi_sent [WMM_AC_NUM]
- uint32_t tx_err
- uint32_t tx_unicast
- uint32_t tx_multicast
- uint32_t tx_broadcast
- uint64_t rx_bytes
- uint32_t rx_pkts
- uint32_t rx_discard
- uint32_t rx_err
- uint32_t rx_unicast
- uint32_t rx_multicast
- uint32_t rx_broadcast
- uint32_t rx_unknown

10.7.1 Detailed Description

This structure is used as a return parameter in the per-interface APIs associated with statistics gathering.

See also

qcsapi_get_interface_stats

10.7.2 Field Documentation

10.7.2.1 tx_bytes

```
uint64_t tx_bytes
```

The number of transmitted bytes on the interface.

10.7.2.2 tx_pkts

```
uint32_t tx_pkts
```

The number of transmitted packets on the interface.

10.7.2.3 tx_discard

```
uint32_t tx_discard
```

The number of discarded transmit packets on the interface.

10.7.2.4 tx_wifi_drop

```
uint32_t tx_wifi_drop[WMM_AC_NUM]
```

The number of dropped data packets failed to transmit through wireless media for each traffic category(TC).

10.7.2.5 tx_wifi_sent

```
uint32_t tx_wifi_sent[WMM_AC_NUM]
```

The number of data packets transmitted through wireless media for each traffic category(TC).

10.7.2.6 tx_err

```
uint32_t tx_err
```

The number of transmit errors on the interface.

10.7.2.7 tx_unicast

```
uint32_t tx_unicast
```

The number of transmitted unicast packets on the interface.

10.7.2.8 tx_multicast

```
uint32_t tx_multicast
```

The number of transmitted multicast packets on the interface.

10.7.2.9 tx_broadcast

```
uint32_t tx_broadcast
```

The number of transmitted broadcast packets on the interface.

10.7.2.10 rx_bytes

```
uint64_t rx_bytes
```

The number of received bytes on the interface.

10.7.2.11 rx_pkts

```
uint32_t rx_pkts
```

The number of received packets on the interface.

10.7.2.12 rx_discard

```
uint32_t rx_discard
```

The number of received packets discarded on the interface.

10.7.2.13 rx_err

```
uint32_t rx_err
```

The number of received packets in error on the interface.

10.7.2.14 rx_unicast

```
uint32_t rx_unicast
```

The number of received unicast packets on the interface.

10.7.2.15 rx_multicast

```
uint32_t rx_multicast
```

The number of received multicast packets on the interface.

10.7.2.16 rx_broadcast

```
uint32_t rx_broadcast
```

The number of received broadcast packets on the interface.

10.7.2.17 rx_unknown

```
uint32_t rx_unknown
```

The number of received unknown packets on the interface.

10.8 _qcsapi_measure_report_result Union Reference

Data Fields

- int common [16]
- struct qcsapi_measure_rpt_tpc_s tpc
- uint8_t basic
- uint8_t cca
- uint8_t rpi [8]
- uint8_t channel_load
- struct qcsapi_measure_rpt_noise_histogram_s noise_histogram
- struct qcsapi_measure_rpt_beacon_s beacon
- struct qcsapi_measure_rpt_frame_s frame
- struct qcsapi_measure_rpt_tran_stream_cat_s tran_stream_cat
- struct qcsapi_measure_rpt_multicast_diag_s multicast_diag
- struct qcsapi_measure_rpt_link_measure_s link_measure
- struct qcsapi_measure_rpt_neighbor_report_s neighbor_report

10.8.1 Detailed Description

Report results for 11h and 11k measurement

See also

qcsapi_wifi_get_node_param

10.8.2 Field Documentation

10.8.2.1 common

```
int common[16]
```

Common place to store results if no specified

10.8.2.2 tpc

```
struct qcsapi_measure_rpt_tpc_s tpc
```

Transmit power control report

10.8.2.3 basic

```
uint8_t basic
```

Basic measurement report

10.8.2.4 cca

```
uint8_t cca
```

CCA measurement report

10.8.2.5 rpi

```
uint8_t rpi[8]
```

RPI measurement report

10.8.2.6 channel_load

```
uint8_t channel_load
```

Channel Load measurement report

10.8.2.7 noise_histogram

```
struct qcsapi_measure_rpt_noise_histogram_s noise_histogram
```

Noise histogram measurement report

10.8.2.8 beacon

```
struct qcsapi_measure_rpt_beacon_s beacon
```

Beacon measurement report

10.8.2.9 frame

```
struct qcsapi_measure_rpt_frame_s frame
```

Frame measurement report

10.8.2.10 tran_stream_cat

```
struct qcsapi_measure_rpt_tran_stream_cat_s tran_stream_cat
```

Transmit stream/category report

10.8.2.11 multicast_diag

```
struct qcsapi_measure_rpt_multicast_diag_s multicast_diag
```

Multicast diagnostics report

10.8.2.12 link_measure

```
struct qcsapi_measure_rpt_link_measure_s link_measure
```

Link measurement

10.8.2.13 neighbor_report

```
struct qcsapi_measure_rpt_neighbor_report_s neighbor_report
```

Neighbor report

10.9 _qcsapi_measure_request_param Union Reference

Data Fields

- struct qcsapi_measure_basic_s basic
- struct qcsapi_measure_cca_s cca
- struct qcsapi_measure_rpi_s rpi
- struct qcsapi_measure_chan_load_s chan_load
- struct qcsapi_measure_noise_his_s noise_his
- struct qcsapi_measure_beacon_s beacon
- struct qcsapi_measure_frame_s frame
- struct qcsapi_measure_tran_steam_cat_s tran_stream_cat
- struct qcsapi_measure_multicast_diag_s multicast_diag

10.9.1 Detailed Description

Request parameter union for 11h and 11k measurement

See also

qcsapi_wifi_get_node_param

10.9.2 Field Documentation

10.9.2.1 basic

```
struct qcsapi_measure_basic_s basic
```

basic measurement paramter

10.9.2.2 cca

`struct qcsapi_measure_cca_s cca`

CCA measurement paramter

10.9.2.3 rpi

`struct qcsapi_measure_rpi_s rpi`

RPI measurement paramter

10.9.2.4 chan_load

`struct qcsapi_measure_chan_load_s chan_load`

Channel Load measurement paramter

10.9.2.5 noise_his

`struct qcsapi_measure_noise_his_s noise_his`

Noise histogram measurement paramter

10.9.2.6 beacon

`struct qcsapi_measure_beacon_s beacon`

Beacon measurement paramter

10.9.2.7 frame

`struct qcsapi_measure_frame_s frame`

Frame measurement paramter

10.9.2.8 tran_stream_cat

`struct qcsapi_measure_tran_steam_cat_s tran_stream_cat`

transmit stream/category measurement

10.9.2.9 multicast_diag

`struct qcsapi_measure_multicast_diag_s multicast_diag`

multicast diagnostics report

10.10 _qcsapi_mlme_stats Struct Reference

Structure containing per client mlme statistics.

Data Fields

- unsigned int **auth**
- unsigned int **auth_fails**
- unsigned int **assoc**
- unsigned int **assoc_fails**
- unsigned int **deauth**
- unsigned int **diassoc**

10.10.1 Detailed Description

This structure is used as a return parameter in the mlme statistics request functions.

See also

qcsapi_wifi_get_mlme_stats_per_association

qcsapi_wifi_get_mlme_stats_per_mac

10.11 _qcsapi_mlme_stats_macs Struct Reference

Structure containing the list of macs.

Data Fields

- qcsapi_mac_addr addr [QCSAPI_MLME_STATS_MAX_MACS]

10.11.1 Detailed Description

This structure is used as a return parameter in mlme statistics macs request function

See also

qcsapi_wifi_get_mlme_stats_macs_list

10.11.2 Field Documentation

10.11.2.1 addr

```
qcsapi_mac_addr addr[QCSAPI_MLME_STATS_MAX_MACS]
```

MAC addresses existing in mlme stats

10.12 _qcsapi_node_txrx_airtime Struct Reference

Data Fields

- qcsapi_mac_addr addr
- uint32_t tx_airtime
- uint32_t tx_airtime_accum
- uint32_t rx_airtime
- uint32_t rx_airtime_accum

10.12.1 Detailed Description

Used with API 'qcsapi_wifi_node_get_txrx_airtime'

10.12.2 Field Documentation

10.12.2.1 addr

```
qcsapi_mac_addr addr
```

MAC address of the node

10.12.2.2 tx_airtime

```
uint32_t tx_airtime
```

current instantaneous airtime

10.12.2.3 tx_airtime_accum

```
uint32_t tx_airtime_accum
```

cumulative airtime during the period from start to stop

10.12.2.4 rx_airtime

```
uint32_t rx_airtime
```

current instantaneous airtime

10.12.2.5 rx_airtime_accum

```
uint32_t rx_airtime_accum
```

cumulative airtime during the period from start to stop

10.13 _qcsapi_phy_stats Struct Reference

Structure containing PHY statistics.

Data Fields

- uint32_t tstamp
- uint32_t assoc
- uint32_t channel
- uint32_t atten
- uint32_t cca_total
- uint32_t cca_tx
- uint32_t cca_rx
- uint32_t cca_int
- uint32_t cca_idle
- uint32_t rx_pkts
- uint32_t rx_gain
- uint32_t rx_cnt_crc
- float rx_noise
- uint32_t tx_pkts
- uint32_t tx_defers
- uint32_t tx_touts
- uint32_t tx_retries
- uint32_t cnt_sp_fail
- uint32_t cnt_lp_fail
- uint32_t last_rx_mcs
- uint32_t last_tx_mcs
- float last_rssi
- float last_rssi_array [QCSAPI_QDRV_NUM_RF_STREAMS]
- float last_rcpi
- float last_evm
- float last_evm_array [QCSAPI_QDRV_NUM_RF_STREAMS]

10.13.1 Detailed Description

This structure is used as a return parameter in the per-interface APIs associated with PHY statistics gathering.

See also

qcsapi_get_phy_stats

10.13.2 Field Documentation

10.13.2.1 tstamp

`uint32_t tstamp`

The timestamp in seconds since system boot up

10.13.2.2 assoc

`uint32_t assoc`

Associated Station count or if Station is associated

10.13.2.3 channel

`uint32_t channel`

Current active channel

10.13.2.4 atten

`uint32_t atten`

Attenuation

10.13.2.5 cca_total

`uint32_t cca_total`

Total CCA

10.13.2.6 cca_tx

`uint32_t cca_tx`

Transmit CCA

10.13.2.7 cca_rx

```
uint32_t cca_rx
```

Receive CCA

10.13.2.8 cca_int

```
uint32_t cca_int
```

CCA interference

10.13.2.9 cca_idle

```
uint32_t cca_idle
```

CCA Idle

10.13.2.10 rx_pkts

```
uint32_t rx_pkts
```

Received packets counter

10.13.2.11 rx_gain

```
uint32_t rx_gain
```

Receive gain in dBm

10.13.2.12 rx_cnt_crc

```
uint32_t rx_cnt_crc
```

Received packet counter with frame check error

10.13.2.13 rx_noise

```
float rx_noise
```

Received noise level in dBm

10.13.2.14 tx_pkts

```
uint32_t tx_pkts
```

Transmitted packets counter

10.13.2.15 tx_defers

```
uint32_t tx_defers
```

Deferred packet counter in transmission

10.13.2.16 tx_touts

```
uint32_t tx_touts
```

Time-out counter for transimitted packets

10.13.2.17 tx_retries

```
uint32_t tx_retries
```

Retried packets counter in transmission

10.13.2.18 cnt_sp_fail

```
uint32_t cnt_sp_fail
```

Counter of short preamble errors

10.13.2.19 cnt_lp_fail

```
uint32_t cnt_lp_fail
```

Counter of long preamble errors

10.13.2.20 last_rx_mcs

```
uint32_t last_rx_mcs
```

MCS index for last received packet

10.13.2.21 last_tx_mcs

```
uint32_t last_tx_mcs
```

MCS index for last transimtted packet

10.13.2.22 last_rssi

```
float last_rssi
```

Received signal strength indicator in dBm

10.13.2.23 last_rssi_array

```
float last_rssi_array[QCSAPI_QDRV_NUM_RF_STREAMS]
```

Per Chain RSSI

10.13.2.24 last_rcpi

```
float last_rcpi
```

Received channel power level in dBm

10.13.2.25 last_evm

```
float last_evm
```

Error vector magnitude measured in dBm

10.13.2.26 last_evm_array

```
float last_evm_array[QCSAPI_QDRV_NUM_RF_STREAMS]
```

Per Chain EVM

10.14 _qcsapi_radar_status Struct Reference

Data Fields

- uint32_t channel
- uint32_t flags
- uint32_t ic_radardetected

10.14.1 Detailed Description

Each channel's Radar status and detected history records

10.14.2 Field Documentation

10.14.2.1 channel

```
uint32_t channel
```

Which channel to be queried. It must be DFS channel.

10.14.2.2 flags

```
uint32_t flags
```

If This API returns without error, it indicates the whether the channel is in non-occupy list currently.

10.14.2.3 ic_radardetected

```
uint32_t ic_radardetected
```

This records times radar signal is detected on this channel.

10.15 _qcsapi_vlan_config Struct Reference

Data Fields

- uint32_t vlan_cfg
- uint32_t member_bitmap [4096/32]
- uint32_t tag_bitmap [4096/32]

10.15.1 Detailed Description

per-interface VLAN configuration.
The definition must be in sync with 'struct qtn_vlan_config' in include/qtn/qtn_vlan.h

10.15.2 Field Documentation

10.15.2.1 vlan_cfg

```
uint32_t vlan_cfg
```

VLAN mode: Access, Trunk, Hybrid, Dynamic or Disabled

10.15.2.2 member_bitmap

```
uint32_t member_bitmap[4096/32]
```

VLAN member bitmap – indicate to which VLANs the interface belongs
Bit X set to 1 in the bitmap indicates the interfaces belongs to VLAN 'X'

10.15.2.3 tag_bitmap

```
uint32_t tag_bitmap[4096/32]
```

VLAN tagging bitmap – indicate on which VLANs should packets be sent with tags on
Bit X set to 1 in the bitmap indicates VLAN 'X' packets are sent with tags on

10.16 qcsapi_ap_properties Struct Reference

This structure represents a set of properties for a single AP.

Data Fields

- qcsapi_SSID ap_name_SSID
- qcsapi_mac_addr ap_mac_addr
- qcsapi_unsigned_int ap_flags
- int ap_channel
- int ap_bw
- int ap_RSSI
- int ap_protocol
- int ap_encryption_modes
- int ap_authentication_mode
- uint32_t ap_best_data_rate
- int ap_wps
- int ap_80211_proto
- int ap_qhop_role
- int ap_beacon_interval
- int ap_dtim_interval
- int ap_is_ess

10.16.1 Detailed Description

This structure represents a set of properties for a single AP.

The contents of this structure can be obtained using the function qcsapi_wifi_get_properties_AP.

This structure is used to return AP scan results.

See also

qcsapi_wifi_get_properties_AP

10.16.2 Field Documentation

10.16.2.1 ap_name_SSID

```
qcsapi_SSID ap_name_SSID
```

The SSID that this AP is using.

10.16.2.2 ap_mac_addr

```
qcsapi_mac_addr ap_mac_addr
```

The MAC address of the wireless interface of the AP.

10.16.2.3 ap_flags

```
qcsapi_unsigned_int ap_flags
```

Flags relevant to the AP. 0 = security disabled, 1 = security enabled

10.16.2.4 ap_channel

```
int ap_channel
```

The operating channel of the AP.

10.16.2.5 ap_bw

```
int ap_bw
```

The max supported bandwidth of the AP.

10.16.2.6 ap_RSSI

```
int ap_RSSI
```

The RSSI of the AP, in the range [0 - 68].

10.16.2.7 ap_protocol

```
int ap_protocol
```

The security protocol in use (none / WPA / WPA2)

10.16.2.8 ap_encryption_modes

```
int ap_encryption_modes
```

The supported encryption modes (eg TKIP, CCMP)

10.16.2.9 ap_authentication_mode

```
int ap_authentication_mode
```

The supported authentication type(s) (eg PSK)

10.16.2.10 ap_best_data_rate

```
uint32_t ap_best_data_rate
```

The fastest data rate this AP is capable of sending

10.16.2.11 ap_wps

```
int ap_wps
```

The capability of WPS.

10.16.2.12 ap_80211_proto

```
int ap_80211_proto
```

The IEEE80211 protocol (e.g. a, b, g, n, ac)

10.16.2.13 ap_qhop_role

```
int ap_qhop_role
```

QHop role (e.g. 0-None, 1-MBS, 2-RBS)

10.16.2.14 ap_beacon_interval

```
int ap_beacon_interval
```

Beacon interval

10.16.2.15 ap_dtim_interval

```
int ap_dtim_interval
```

DTIM interval

10.16.2.16 ap_is_ess

```
int ap_is_ess
```

Operating mode is infrastructure(ess) or adhoc(ibss)

10.17 qcsapi_assoc_records Struct Reference

Data Fields

- qcsapi_mac_addr addr [QCSAPI_ASSOC_MAX_RECORDS]
- uint32_t timestamp [QCSAPI_ASSOC_MAX_RECORDS]

10.17.1 Detailed Description

Used with API 'qcsapi_wifi_get_assoc_records'

10.17.2 Field Documentation

10.17.2.1 addr

```
qcsapi_mac_addr addr[QCSAPI_ASSOC_MAX_RECORDS]
```

MAC addresses of remote nodes that have associated

10.17.2.2 timestamp

```
uint32_t timestamp[QCSAPI_ASSOC_MAX_RECORDS]
```

Time stamp of the most recent association by the corresponding remote node

10.18 qcsapi_autochan_rpt Struct Reference

Structure containing auto channel report for initial channel selection.

Data Fields

- uint8_t num
- uint8_t chan [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t dfs [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t txpwr [QCSAPI_SCS_REPORT_CHAN_NUM]
- int32_t metric [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t numbeacons [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t cci [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t aci [QCSAPI_SCS_REPORT_CHAN_NUM]

10.18.1 Detailed Description

This structure is used as a return parameter in the Auto Channel API to return report for initial channel selection.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_autochan_report

10.18.2 Field Documentation

10.18.2.1 num

```
uint8_t num
```

Valid record number in the following attribute arrays

10.18.2.2 chan

```
uint8_t chan[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Channel number

10.18.2.3 dfs

```
uint8_t dfs[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Whether channel is DFS channel or not

10.18.2.4 txpwr

```
uint8_t txpwr[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Txpower

10.18.2.5 metric

```
int32_t metric[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Ranking metric

10.18.2.6 numbeacons

```
uint32_t numbeacons[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Number of beacons detected

10.18.2.7 cci

```
uint32_t cci[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Co-channel interference index

10.18.2.8 aci

```
uint32_t aci[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Adjacent Channel interference index

10.19 qcsapi_cca_info Struct Reference

Data Fields

- int cca_channel
- int cca_duration

10.19.1 Detailed Description

Used with API 'qcsapi_wifi_start_cca'

10.19.2 Field Documentation

10.19.2.1 cca_channel

```
int cca_channel
```

Channel to switch to for off channel CCA measurements

10.19.2.2 cca_duration

```
int cca_duration
```

Duration to stay on the channel being measured, in milliseconds

10.20 qcsapi_channel_power_table Struct Reference

Structure to contain the power table for a single channel.

Data Fields

- uint8_t channel
- int power_20M [QCSAPI_POWER_TOTAL]
- int power_40M [QCSAPI_POWER_TOTAL]
- int power_80M [QCSAPI_POWER_TOTAL]

10.20.1 Detailed Description

This structure is used as an input or return parameter in the channel power table APIs. It is filled in as as a power level (in dBm) for each combination of channel bandwidth (20, 40, 80MHz), spatial stream (1, 2, 3, 4) and beamforming on vs. off. A total of 24 power levels must be configured.

For example, the following code snippet shows an initialisation of the structure and the corresponding channel/bandwidth/SS power levels.

```
qcsapi_channel_power_table channel_36;
memset(&channel_36, 0, sizeof(channel_36));
channel_36.channel = 36;
channel_36.power_20M[QCSAPI_POWER_INDEX_BFOFF_1SS] = 19;
channel_36.power_20M[QCSAPI_POWER_INDEX_BFOFF_2SS] = 19;
...
channel_36.power_40M[QCSAPI_POWER_INDEX_BFON_3SS] = 17;
channel_36.power_40M[QCSAPI_POWER_INDEX_BFON_4SS] = 17;
...
channel_36.power_80M[QCSAPI_POWER_INDEX_BFON_3SS] = 15;
channel_36.power_80M[QCSAPI_POWER_INDEX_BFON_4SS] = 15;
```

See also

qcsapi_wifi_get_chan_power_table

qcsapi_wifi_set_chan_power_table

qcsapi_power_indices

10.20.2 Field Documentation

10.20.2.1 channel

```
uint8_t channel
```

The channel number.

10.20.2.2 power_20M

```
int power_20M[QCSAPI_POWER_TOTAL]
```

The power for 20Mhz bandwidth. For the index, please see the definition for "QCSAPI_POWER_INDEX..."

10.20.2.3 power_40M

```
int power_40M[QCSAPI_POWER_TOTAL]
```

The power for 40Mhz bandwidth. For the index, please see the definition for "QCSAPI_POWER_INDEX..."

10.20.2.4 power_80M

```
int power_80M[QCSAPI_POWER_TOTAL]
```

The power for 80Mhz bandwidth. For the index, please see the definition for "QCSAPI_POWER_INDEX..."

10.21 qcsapi_data_128bytes Struct Reference

Convenience definition to represent a 128 unsigned byte array.

Data Fields

- uint8_t **data** [128]

10.21.1 Detailed Description

Convenience definition to represent a 128 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 128 byte array. This type is not RPC endian-safe and should not be used to pass integer values.

10.22 qcsapi_data_1Kbytes Struct Reference

Convenience definition to represent a 1024 unsigned byte array.

Data Fields

- uint8_t **data** [1024]

10.22.1 Detailed Description

Convenience definition to represent a 1024 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 1024 byte array.

This type is not RPC endian-safe and should not be used to pass integer values.

10.23 qcsapi_data_256bytes Struct Reference

Convenience definition to represent a 256 unsigned byte array.

Data Fields

- uint8_t **data** [256]

10.23.1 Detailed Description

Convenience definition to represent a 256 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 256 byte array.

This type is not RPC endian-safe and should not be used to pass integer values.

10.24 qcsapi_data_2Kbytes Struct Reference

Convenience definition to represent a 2048 unsigned byte array.

Data Fields

- uint8_t **data** [2048]

10.24.1 Detailed Description

Convenience definition to represent a 2048 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 2048 byte array.

This type is not RPC endian-safe and should not be used to pass integer values.

10.25 qcsapi_data_3Kbytes Struct Reference

Convenience definition to represent a 3072 unsigned byte array.

Data Fields

- uint8_t **data** [3072]

10.25.1 Detailed Description

Convenience definition to represent a 3072 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 3072 byte array.

This type is not RPC endian-safe and should not be used to pass integer values.

10.26 qcsapi_data_4Kbytes Struct Reference

Convenience definition to represent a 4096 unsigned byte array.

Data Fields

- uint8_t **data** [4096]

10.26.1 Detailed Description

Convenience definition to represent a 4096 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 4096 byte array.

This type is not RPC endian-safe and should not be used to pass integer values.

10.27 qcsapi_data_512bytes Struct Reference

Convenience definition to represent a 512 unsigned byte array.

Data Fields

- uint8_t **data** [512]

10.27.1 Detailed Description

Convenience definition to represent a 512 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 512 byte array. This type is not RPC endian-safe and should not be used to pass integer values.

10.28 qcsapi_data_64bytes Struct Reference

Convenience definition to represent a 64 byte array.

Data Fields

- uint8_t **data** [64]

10.28.1 Detailed Description

Convenience definition to represent a 64 byte array.

Note

This type should not be considered a string as embedded NULL bytes are allowed as part of a 64 byte array. This type is not RPC endian-safe and should not be used to pass integer values.

10.29 qcsapi_int_array1024 Struct Reference

Convenience definition to represent a 1024 integer array.

Data Fields

- int32_t **val** [1024]

10.29.1 Detailed Description

Convenience definition to represent a 1024 integer array.

Note

This type is RPC endian-safe and can be used to pass structures that are not pure byte sequences.

10.30 qcsapi_int_array128 Struct Reference

Convenience definition to represent a 128 integer array.

Data Fields

- int32_t **val** [128]

10.30.1 Detailed Description

Convenience definition to represent a 128 integer array.

Note

This type is RPC endian-safe and can be used to pass structures that are not pure byte sequences.

10.31 qcsapi_int_array256 Struct Reference

Convenience definition to represent a 256 integer array.

Data Fields

- int32_t **val** [256]

10.31.1 Detailed Description

Convenience definition to represent a 256 integer array.

Note

This type is RPC endian-safe and can be used to pass structures that are not pure byte sequences.

10.32 qcsapi_int_array32 Struct Reference

Convenience definition to represent a 32 integer array.

Data Fields

- int32_t **val** [32]

10.32.1 Detailed Description

Convenience definition to represent a 32 integer array.

Note

This type is RPC endian-safe and can be used to pass structures that are not pure byte sequences.

10.33 qcsapi_int_array768 Struct Reference

Convenience definition to represent a 768 integer array.

Data Fields

- int32_t **val** [768]

10.33.1 Detailed Description

Convenience definition to represent a 768 integer array.

Note

This type is RPC endian-safe and can be used to pass structures that are not pure byte sequences.

10.34 qcsapi_log_param Struct Reference

Struct to store parameters for respective module.

Data Fields

- string_32 name
- string_32 value

10.34.1 Field Documentation

10.34.1.1 name

```
string_32 name
```

The name of the parameters e.g. level, module.

10.34.1.2 value

```
string_32 value
```

The name specific value.

10.35 qcsapi_mac_list Struct Reference

Data Fields

- uint32_t flags
- uint32_t num_entries
- uint8_t macaddr [QCSAPI_MAX_MACS_SIZE]

10.35.1 Detailed Description

Structure to contain the results for a call to qcsapi_get_client_mac_list

See also

qcsapi_get_client_mac_list

10.35.2 Field Documentation

10.35.2.1 flags

```
uint32_t flags
```

Flags for the resulting list returned.

- 0x1 - The returned addresses are behind a four address node.
- 0x2 - The returned address list is truncated.

10.35.2.2 num_entries

```
uint32_t num_entries
```

The number of entries returned in the MAC address array.

10.35.2.3 macaddr

```
uint8_t macaddr[QCSAPI_MAX_MACS_SIZE]
```

The array of MAC addresses behind the node.

10.36 qcsapi_measure_basic_s Struct Reference

Data Fields

- uint16_t offset
- uint16_t duration
- uint8_t channel

10.36.1 Detailed Description

Basic measurement parameter

See also

_qcsapi_measure_request_param

10.36.2 Field Documentation

10.36.2.1 offset

```
uint16_t offset
```

offset to start measurement, based on microsecond

10.36.2.2 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.36.2.3 channel

```
uint8_t channel
```

channel to execute the measurement, based on IEEEE channel number

10.37 qcsapi_measure_beacon_s Struct Reference

Data Fields

- uint8_t op_class
- uint8_t channel
- uint16_t duration
- uint8_t mode
- uint8_t bssid [6]

10.37.1 Detailed Description

Beacon measurement parameter

See also

_qcsapi_measure_request_param

10.37.2 Field Documentation

10.37.2.1 op_class

```
uint8_t op_class
```

operating class, with channel and region to decide which frequency to execute

10.37.2.2 channel

```
uint8_t channel
```

IEEE channel number, with operating class and region to decide which frequency to execute

10.37.2.3 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.37.2.4 mode

```
uint8_t mode
```

beacon measurement mode, 0 passive, 1 active, 2 table

10.37.2.5 bssid

```
uint8_t bssid[6]
```

specified bssid for beacon measurement

10.38 qcsapi_measure_cca_s Struct Reference

Data Fields

- uint16_t offset
- uint16_t duration
- uint8_t channel

10.38.1 Detailed Description

CCA measurement parameter

See also

_qcsapi_measure_request_param

10.38.2 Field Documentation

10.38.2.1 offset

```
uint16_t offset
```

offset to start measurement, based on microsecond

10.38.2.2 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.38.2.3 channel

```
uint8_t channel
```

channel to execute the measurement, based on IEEEE channel number

10.39 qcsapi_measure_chan_load_s Struct Reference

Data Fields

- uint8_t op_class
- uint8_t channel
- uint16_t duration

10.39.1 Detailed Description

Channel Load measurement parameter

See also

_qcsapi_measure_request_param

10.39.2 Field Documentation

10.39.2.1 op_class

```
uint8_t op_class
```

operating class, with channel and region to decide which frequency to execute

10.39.2.2 channel

```
uint8_t channel
```

IEEE channel number, with operating class and region to decide which frequency to execute

10.39.2.3 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.40 qcsapi_measure_frame_s Struct Reference

Data Fields

- uint8_t op_class
- uint8_t channel
- uint16_t duration
- uint8_t type
- uint8_t mac_address [6]

10.40.1 Detailed Description

Frame measurement parameter

See also

_qcsapi_measure_request_param

10.40.2 Field Documentation

10.40.2.1 op_class

```
uint8_t op_class
```

operating class, with channel and region to decide which frequency to execute

10.40.2.2 channel

```
uint8_t channel
```

IEEE channel number, with operating class and region to decide which frequency to execute

10.40.2.3 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.40.2.4 type

```
uint8_t type
```

frame type, currently only frame count report(1) is supported

10.40.2.5 mac_address

```
uint8_t mac_address[6]
```

specified mac address for frame measurement

10.41 qcsapi_measure_multicast_diag_s Struct Reference

Data Fields

- uint16_t duration
- uint8_t group_mac [6]

10.41.1 Detailed Description

Multicast diagnostics report parameter

See also

_qcsapi_measure_request_param

10.41.2 Field Documentation

10.41.2.1 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.41.2.2 group_mac

```
uint8_t group_mac[6]
```

specified group mac_address for measurement

10.42 qcsapi_measure_neighbor_item_s Struct Reference

Data Fields

- uint8_t **bssid** [6]
- uint32_t **bssid_info**
- uint8_t **operating_class**
- uint8_t **channel**
- uint8_t **phy_type**

10.42.1 Detailed Description

Neighbor report item

See also

_qcsapi_measure_report_result

10.43 qcsapi_measure_noise_his_s Struct Reference

Data Fields

- uint8_t op_class
- uint8_t channel
- uint16_t duration

10.43.1 Detailed Description

Noise histogram measurement parameter

See also

_qcsapi_measure_request_param

10.43.2 Field Documentation

10.43.2.1 op_class

```
uint8_t op_class
```

operating class, with channel and region to decide which frequency to execute

10.43.2.2 channel

```
uint8_t channel
```

IEEE channel number, with operating class and region to decide which frequency to execute

10.43.2.3 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.44 qcsapi_measure_rpi_s Struct Reference

Data Fields

- uint16_t offset
- uint16_t duration
- uint8_t channel

10.44.1 Detailed Description

RPI measurement parameter

See also

_qcsapi_measure_request_param

10.44.2 Field Documentation

10.44.2.1 offset

`uint16_t offset`

offset to start measurement, based on microsecond

10.44.2.2 duration

`uint16_t duration`

duration to do the measurement, based on microsecond

10.44.2.3 channel

`uint8_t channel`

channel to execute the measurement, based on IEEEE channel number

10.45 qcsapi_measure_rpt_beacon_s Struct Reference

Data Fields

- uint8_t **rep_frame_info**
- uint8_t **rcpi**
- uint8_t **rsni**
- uint8_t **bssid** [6]
- uint8_t **antenna_id**
- uint32_t **parent_tsf**

10.45.1 Detailed Description

Beacon measurement report

See also

_qcsapi_measure_report_result

10.46 qcsapi_measure_rpt_frame_s Struct Reference

Data Fields

- uint32_t **sub_ele_report**
- uint8_t **ta** [6]
- uint8_t **bssid** [6]
- uint8_t **phy_type**
- uint8_t **avg_rcpi**
- uint8_t **last_rsni**
- uint8_t **last_rcpi**
- uint8_t **antenna_id**
- uint16_t **frame_count**

10.46.1 Detailed Description

Frame measurement report

See also

_qcsapi_measure_report_result

10.47 qcsapi_measure_rpt_link_measure_s Struct Reference

Data Fields

- struct qcsapi_measure_rpt_tpc_report_s **tpc_report**
- uint8_t **recv_antenna_id**
- uint8_t **tran_antenna_id**
- uint8_t **rcpi**
- uint8_t **rsni**

10.47.1 Detailed Description

Link measurement report

See also

_qcsapi_measure_report_result

10.48 qcsapi_measure_rpt_multicast_diag_s Struct Reference

Data Fields

- uint8_t **reason**
- uint32_t **mul_rec_msdu_cnt**
- uint16_t **first_seq_num**
- uint16_t **last_seq_num**
- uint16_t **mul_rate**

10.48.1 Detailed Description

Multicast diagnostics report

See also

_qcsapi_measure_report_result

10.49 qcsapi_measure_rpt_neighbor_report_s Struct Reference

Data Fields

- uint8_t **item_num**
- struct qcsapi_measure_neighbor_item_s **items** [3]

10.49.1 Detailed Description

Neighbor measurement report

See also

_qcsapi_measure_report_result

10.50 qcsapi_measure_rpt_noise_histogram_s Struct Reference

Data Fields

- uint8_t **antenna_id**
- uint8_t **anpi**
- uint8_t **ipi** [11]

10.50.1 Detailed Description

Noise histogram measurement report

See also

_qcsapi_measure_report_result

10.51 qcsapi_measure_rpt_tpc_report_s Struct Reference

Data Fields

- int8_t **tx_power**
- int8_t **link_margin**

10.51.1 Detailed Description

TPC measurement report

See also

_qcsapi_measure_report_result

10.52 qcsapi_measure_rpt_tpc_s Struct Reference

Data Fields

- int8_t **link_margin**
- int8_t **tx_power**

10.52.1 Detailed Description

Transmit power control report

See also

_qcsapi_measure_report_result

10.53 qcsapi_measure_rpt_tran_stream_cat_s Struct Reference

Data Fields

- uint8_t **reason**
- uint32_t **tran_msdu_cnt**
- uint32_t **msdu_discard_cnt**
- uint32_t **msdu_fail_cnt**
- uint32_t **msdu_mul_retry_cnt**
- uint32_t **qos_lost_cnt**
- uint32_t **avg_queue_delay**
- uint32_t **avg_tran_delay**
- uint8_t **bin0_range**
- uint32_t **bins** [6]

10.53.1 Detailed Description

Transmit stream/category report

See also

_qcsapi_measure_report_result

10.54 qcsapi_measure_tran_steam_cat_s Struct Reference

Data Fields

- uint16_t duration
- uint8_t peer_sta [6]
- uint8_t tid
- uint8_t bin0

10.54.1 Detailed Description

Transmit stream/category measurement parameter

See also

_qcsapi_measure_request_param

10.54.2 Field Documentation

10.54.2.1 duration

```
uint16_t duration
```

duration to do the measurement, based on microsecond

10.54.2.2 peer_sta

```
uint8_t peer_sta[6]
```

specified mac_address for measurement

10.54.2.3 tid

```
uint8_t tid
```

traffic ID

10.54.2.4 bin0

```
uint8_t bin0
```

bin 0

10.55 qcsapi_node_stats Struct Reference

Structure to contain per node statistics.

Data Fields

- uint64_t tx_bytes
- uint32_t tx_pkts
- uint32_t tx_discard
- uint32_t tx_wifi_drop [WMM_AC_NUM]
- uint32_t tx_err
- uint32_t tx_unicast
- uint32_t tx_multicast
- uint32_t tx_broadcast
- uint32_t tx_phy_rate
- uint64_t rx_bytes
- uint32_t rx_pkts
- uint32_t rx_discard
- uint32_t rx_err
- uint32_t rx_unicast
- uint32_t rx_multicast
- uint32_t rx_broadcast
- uint32_t rx_unknown
- uint32_t rx_phy_rate
- qcsapi_mac_addr mac_addr
- int32_t hw_noise
- int32_t snr
- int32_t rssi
- int32_t bw

10.55.1 Detailed Description

This structure is used as a return parameter in the per-node association APIs associated with statistics gathering.

See also

qcsapi_wifi_get_node_stats

10.55.2 Field Documentation

10.55.2.1 tx_bytes

```
uint64_t tx_bytes
```

The number of transmitted bytes to the node.

10.55.2.2 tx_pkts

```
uint32_t tx_pkts
```

The number of transmitted packets to the node.

10.55.2.3 tx_discard

```
uint32_t tx_discard
```

The number of transmit discards to the node.

10.55.2.4 tx_wifi_drop

```
uint32_t tx_wifi_drop[WMM_AC_NUM]
```

The number of dropped data packets failed to transmit through wireless media for each traffic category(TC).

10.55.2.5 tx_err

```
uint32_t tx_err
```

The number of transmit errors to the node.

10.55.2.6 tx_unicast

```
uint32_t tx_unicast
```

The number of transmitted unicast packets to the node.

10.55.2.7 tx_multicast

```
uint32_t tx_multicast
```

The number of transmitted multicast packets to the node.

10.55.2.8 tx_broadcast

```
uint32_t tx_broadcast
```

The number of transmitted broadcast packets to the node.

10.55.2.9 tx_phy_rate

```
uint32_t tx_phy_rate
```

TX PHY rate in megabits per second (MBPS)

10.55.2.10 rx_bytes

```
uint64_t rx_bytes
```

The number of received bytes from the node.

10.55.2.11 rx_pkts

```
uint32_t rx_pkts
```

The number of received packets from the node.

10.55.2.12 rx_discard

```
uint32_t rx_discard
```

The numbder of received packets discarded from the node.

10.55.2.13 rx_err

```
uint32_t rx_err
```

The number of received packets in error from the node.

10.55.2.14 rx_unicast

```
uint32_t rx_unicast
```

The number of received unicast packets from the node.

10.55.2.15 rx_multicast

```
uint32_t rx_multicast
```

The number of received multicast packets from the node.

10.55.2.16 rx_broadcast

```
uint32_t rx_broadcast
```

The number of received broadcast packets form the node.

10.55.2.17 rx_unknown

```
uint32_t rx_unknown
```

The number of received unknown packets from the node.

10.55.2.18 rx_phy_rate

```
uint32_t rx_phy_rate
```

RX PHY rate in megabits per second (MBPS)

10.55.2.19 mac_addr

```
qcsapi_mac_addr mac_addr
```

The MAC address of the node.

10.55.2.20 hw_noise

```
int32_t hw_noise
```

The hw noise of the node.

10.55.2.21 snr

```
int32_t snr
```

The snr of the node.

10.55.2.22 rssi

```
int32_t rssi
```

The rssi of the node.

10.55.2.23 bw

```
int32_t bw
```

The bandwidth of the node.

10.56 qcsapi_sample_assoc_data Struct Reference

Data Fields

- qcsapi_mac_addr **mac_addr**
- uint8_t **assoc_id**
- uint8_t **bw**
- uint8_t **tx_stream**
- uint8_t **rx_stream**
- uint32_t **time_associated**
- uint32_t **achievable_tx_phy_rate**
- uint32_t **achievable_rx_phy_rate**
- uint32_t **rx_packets**
- uint32_t **tx_packets**
- uint32_t **rx_errors**
- uint32_t **tx_errors**
- uint32_t **rx_dropped**
- uint32_t **tx_dropped**
- uint32_t **tx_wifi_drop** [WMM_AC_NUM]
- uint32_t **rx_ucast**
- uint32_t **tx_ucast**
- uint32_t **rx_mcast**
- uint32_t **tx_mcast**
- uint32_t **rx_bcast**
- uint32_t **tx_bcast**
- uint16_t **link_quality**
- uint32_t **ip_addr**
- uint64_t **rx_bytes**
- uint64_t **tx_bytes**
- uint32_t **last_rssi_dbm** [QCSAPI_NUM_ANT]
- uint32_t **last_rcpi_dbm** [QCSAPI_NUM_ANT]
- uint32_t **last_evm_dbm** [QCSAPI_NUM_ANT]
- uint32_t **last_hw_noise** [QCSAPI_NUM_ANT]
- uint8_t **protocol**
- uint8_t **vendor**

10.57 qcsapi_scs_currchan_rpt Struct Reference

Structure containing SCS report for current channel.

Data Fields

- uint8_t chan
- uint16_t cca_try
- uint16_t cca_idle
- uint16_t cca_busy
- uint16_t cca_intf
- uint16_t cca_tx
- uint16_t tx_ms
- uint16_t rx_ms
- uint32_t pmbl

10.57.1 Detailed Description

This structure is used as a return parameter in the SCS API to return report for the current channel.

See also

qcsapi_wifi_get_scs_currchan_report

10.57.2 Field Documentation

10.57.2.1 chan

```
uint8_t chan
```

Current channel number

10.57.2.2 cca_try

```
uint16_t cca_try
```

Total try count for cca sampling

10.57.2.3 cca_idle

```
uint16_t cca_idle
```

CCA idle count

10.57.2.4 cca_busy

```
uint16_t cca_busy
```

CCA busy count

10.57.2.5 cca_intf

```
uint16_t cca_intf
```

CCA interference count

10.57.2.6 cca_tx

```
uint16_t cca_tx
```

CCA transmitting count

10.57.2.7 tx_ms

```
uint16_t tx_ms
```

Transmiting time in ms

10.57.2.8 rx_ms

```
uint16_t rx_ms
```

Receiving time in ms

10.57.2.9 pmbl

```
uint32_t pmbl
```

Preamble error count

10.58 qcsapi_scs_param_rpt Struct Reference

Data Fields

- uint32_t **scs_cfg_param**
- uint32_t **scs_signed_param_flag**

10.58.1 Detailed Description

This structure is the same as 'struct ieee80211req_scs_param_rpt', but (re)defined for convenience

10.59 qcsapi_scs_ranking_rpt Struct Reference

Structure containing SCS report for all channels.

Data Fields

- uint8_t num
- uint8_t chan [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t dfs [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t txpwr [QCSAPI_SCS_REPORT_CHAN_NUM]
- int32_t metric [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t metric_age [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint16_t cca_intf [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t pmbl_ap [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t pmbl_sta [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t duration [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint32_t times [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t chan_avail_status [QCSAPI_SCS_REPORT_CHAN_NUM]

10.59.1 Detailed Description

This structure is used as a return parameter in the SCS API to return report for the all channels.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_scs_stat_report

10.59.2 Field Documentation

10.59.2.1 num

```
uint8_t num
```

Valid record number in the following attribute arrays

10.59.2.2 chan

```
uint8_t chan[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Channel numbers

10.59.2.3 dfs

```
uint8_t dfs[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Whether channel is DFS channel or not

10.59.2.4 txpwr

```
uint8_t txpwr[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Txpower

10.59.2.5 metric

```
int32_t metric[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Ranking metric

10.59.2.6 metric_age

```
uint32_t metric_age[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Ranking metric age

10.59.2.7 cca_intf

```
uint16_t cca_intf[QCSAPI_SCS_REPORT_CHAN_NUM]
```

CCA interference

10.59.2.8 pmbl_ap

```
uint32_t pmbl_ap[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Preamble error detected by AP

10.59.2.9 pmbl_sta

```
uint32_t pmbl_sta[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Maximum preamble error detected by STAs

10.59.2.10 duration

```
uint32_t duration[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Amount of time the channel was used in seconds

10.59.2.11 times

```
uint32_t times[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Number of times the channel was used

10.59.2.12 chan_avail_status

```
uint8_t chan_avail_status[QCSAPI_SCS_REPORT_CHAN_NUM]
```

Channel availability status

10.60 qcsapi_scs_score_rpt Struct Reference

Structure containing the scores of all channels.

Data Fields

- uint8_t **num**
- uint8_t **chan** [QCSAPI_SCS_REPORT_CHAN_NUM]
- uint8_t **score** [QCSAPI_SCS_REPORT_CHAN_NUM]

10.60.1 Detailed Description

This structure is used as a return parameter in the SCS API to return the scores of the all channels.

The attributes for a certain channel use the same index into each attribute array.

See also

qcsapi_wifi_get_scs_score_report

10.61 qvsp_cfg_param Struct Reference

Data Fields

- const char * **name**
- const char * **desc**
- const char * **units**
- uint32_t **default_val**
- uint32_t **min_val**
- uint32_t **max_val**

10.62 qvsp_hash_flds Union Reference

Data Fields

- struct qvsp_hash_flds_ipv4 **ipv4**
- struct qvsp_hash_flds_ipv6 **ipv6**

10.63 qvsp_hash_flds_ipv4 Struct Reference

Data Fields

- __be32 saddr
- __be32 daddr
- __be16 sport
- __be16 dport

10.63.1 Detailed Description

Defines a stream based on source and destination

10.63.2 Field Documentation

10.63.2.1 saddr

```
__be32 saddr
```

IP source address

10.63.2.2 daddr

```
__be32 daddr
```

IP destination address

10.63.2.3 sport

```
__be16 sport
```

UDP/TCP source port

10.63.2.4 dport

```
__be16 dport
```

UDP/TCP destination port

10.64 qvsp_hash_flds_ipv6 Struct Reference

Data Fields

- struct in6_addr saddr
- struct in6_addr daddr
- __be16 sport
- __be16 dport

10.64.1 Field Documentation

10.64.1.1 saddr

```
struct in6_addr saddr
```

IP source address

10.64.1.2 daddr

```
struct in6_addr daddr
```

IP destination address

10.64.1.3 sport

```
__be16 sport
```

UDP/TCP source port

10.64.1.4 dport

```
__be16 dport
```

UDP/TCP destination port

10.65 qvsp_rule_flds Struct Reference

Data Fields

- uint32_t **param** [QVSP_RULE_PARAM_MAX]

10.66 qvsp_rule_param Struct Reference

Data Fields

- const char * **name**
- const char * **desc**
- const char * **units**
- uint32_t **min_val**
- uint32_t **max_val**

10.67 qvsp_stats Struct Reference

Data Fields

- uint32_t **is_qtm**
- uint32_t **strm_enable**
- uint32_t **strm_disable**
- uint32_t **strm_disable_remote**
- uint32_t **strm_reenable**
- uint32_t **fat_over**
- uint32_t **fat_under**
- uint32_t **fat_chk_disable**
- uint32_t **fat_chk_reenable**
- uint32_t **fat_chk_squeeze**
- uint32_t **fat_chk_loosen**
- struct qvsp_stats_if **stats_if** [QVSP_IF_MAX]

10.68 qvsp_stats_if Struct Reference

Data Fields

- uint32_t **strm_add**
- uint32_t **strm_none**
- uint32_t **pkt_chk**
- uint32_t **pkt_tcp**
- uint32_t **pkt_udp**
- uint32_t **pkt_other**
- uint32_t **pkt_ignore**
- uint32_t **pkt_sent**
- uint32_t **pkt_drop_throttle**
- uint32_t **pkt_drop_disabled**
- uint32_t **pkt_demoted**
- uint32_t **pkt_frag_found**
- uint32_t **pkt_frag_not_found**

10.69 qvsp_strm_info Struct Reference

Data Fields

- union qvsp_hash_flds **hash_flds**
- uint16_t **node_idx**
- uint8_t **node_mac** [6]
- uint8_t **vap_pri**
- uint8_t **tid**
- uint16_t **hairpin_id**
- uint16_t **hairpin_type**
- uint8_t **ip_version**
- uint8_t **ip_proto**
- uint8_t **ac_in**
- uint8_t **ac_out**
- uint8_t **strm_state**
- uint8_t **disable_remote**
- uint8_t **is_3rdpt_udp_us**
- uint16_t **last_ref_secs**
- uint32_t **ni_inv_phy_rate**
- uint32_t **phy_rate_disabled**
- uint32_t **bytes_max**
- uint32_t **ni_cost**
- uint16_t **cost_current**
- uint16_t **cost_max**
- uint8_t **hash**
- uint8_t **dir**
- uint32_t **throt_policy**
- uint32_t **throt_rate**
- uint32_t **demote_rule**
- uint32_t **demote_state**
- struct qvsp_strm_stats **prev_stats**

10.70 qvsp_strm_info_safe Struct Reference

Data Fields

- uint16_t **node_idx**
- uint8_t **node_mac** [6]
- uint8_t **vap_pri**
- uint8_t **tid**
- uint16_t **hairpin_id**
- uint16_t **hairpin_type**
- uint8_t **ac_in**
- uint8_t **ac_out**
- uint8_t **strm_state**
- uint8_t **disable_remote**
- uint8_t **is_3rdpt_udp_us**
- uint16_t **last_ref_secs**
- uint32_t **ni_inv_phy_rate**
- uint32_t **phy_rate_disabled**
- uint32_t **bytes_max**
- uint32_t **ni_cost**
- uint16_t **cost_current**
- uint16_t **cost_max**
- uint8_t **hash**
- uint8_t **dir**
- uint32_t **throt_policy**
- uint32_t **throt_rate**
- uint32_t **demote_rule**
- uint32_t **demote_state**
- struct qvsp_strm_stats **prev_stats**

10.71 qvsp_strm_stats Struct Reference

Data Fields

- unsigned long **first_ref**
- uint32_t **pkts**
- uint32_t **bytes**
- uint32_t **bytes_sent**
- uint32_t **pkts_sent**

10.72 qvsp_strms Struct Reference

Data Fields

- struct qvsp_strm_info_safe **strms** [QVSP_STRM_MAX_ENTRIES]

10.73 qvsp_wl_flds Struct Reference

Data Fields

- union qvsp_hash_flds **hflds**
- uint8_t s_cidr_bits
- uint8_t d_cidr_bits
- uint8_t ip_version

10.73.1 Detailed Description

Whitelist definition. Passing streams are compared with the stream defined in 'hflds', ANDed with netmasks

10.73.2 Field Documentation

10.73.2.1 s_cidr_bits

```
uint8_t s_cidr_bits
```

IP source CIDR bitcount

10.73.2.2 d_cidr_bits

```
uint8_t d_cidr_bits
```

IP destination CIDR bitcount

10.73.2.3 ip_version

```
uint8_t ip_version
```

IP version