Qualcomm

Qualcomm Technologies, Inc.

QCC730 API Specification

API Reference

80-Y9730-4 Rev. AH

March 25, 2026

Revision History

Revision	Date	Description
AA	November 2023	Initial release
AB	February 2024	Added Firmware Upgrade APIs. Other miscellaneous updates.
AC	April 2024	Added qapi_Flash_Get_Info and updated Module IDs description
AD	July 2024	Added Fatal Error Manager, GPIO module, HFC APIs, HTTP Client, Networking Status, and UART sections.
AE	March 2025	No technical content was changed in this revision; editioral changes only.
AF	March 2025	Added PRNG, RRAM, and RTC APIs.
AG	May 2025	Added HTTP Server section.
AH	March 2026	Numerous updates throughout document.

Contents

1 Introduction

1.1 Purpose

This document provides the QCC730 API Specification.

1.2 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands and command variables appear in a different font, for example, **`copy a:*.* b:`**.

1.3 Technical Assistance

For assistance or clarification on information in this document, open a technical support case at https://support.qualcomm.com.

You will need to register for a Qualcomm ID account and your company must have support enabled to access our Case system.

Other systems and support resources are listed on https://support.qualcomm.com.

If you need further assistance, you can send an email to qualcomm.support@qti.qualcomm.com.

2 Functional overview

The QAPI provides access to low-level system software, exposing:

- System on-chip (SoC) hardware/peripherals
- Input/output (I/O) interfaces
- Communication interfaces
- Security services
- Hardware/software configuration and power management interfaces
- All other associated functionality deemed part of a chip support package

The QAPI enhances the portability of application-specific OEM software by abstracting it from low-level system software provided by Qualcomm.

3 Console module

3.1 Console APIs

3.1.1 Data Structure Documentation

3.1.1.1 struct QAPI_Console_Parameter_s

Console parameters.

Data fields

Type	Parameter	Description
char *	String_Value	
int32_t	Integer_Value	
qbool_t	Integer_Is_-Valid	

3.1.1.2 struct QAPI_Console_Command_s

QAPI console command.

Data fields

Type	Parameter	Description
QAPI_-Concole_-Command_-Function_t	Command_-Function	Function called when the command is executed from the CLI.
const char *	Command_-String	String representation of the function.
const char *	Usage_String	Usage string for the command.
const char *	Description	Description string for the commmand.

3.1.1.3 struct QAPI_Console_Command_Group_s

QAPI console command group.

Data fields

Type	Parameter	Description
const char *	Group_String	String representation of the group.
uint32_t	Command_-Count	Number of commands in the group.

Type	Parameter	Description
const QAPI_Console_Command_t *	Command_List	List of commands for the group.

3.1.2 Typedef Documentation

3.1.2.1 typedef void* QAPI_Console_Handle_t

Console handle.

3.1.2.2 typedef void* QAPI_Console_Group_Handle_t

Console group handle.

3.1.2.3 typedef struct QAPI_Console_Parameter_s QAPI_Console_Parameter_t

Console parameters.

3.1.2.4 typedef qapi_Status_t(* QAPI_Concole_Command_Function_t)(uint32_t Parameter_Count, QAPI_Console_Parameter_t *Parameter_List)

QAPI status.

3.1.2.5 typedef struct QAPI_Console_Command_s QAPI_Console_Command_t

QAPI console command.

3.1.2.6 typedef struct QAPI_Console_Command_Group_s QAPI_Console_Command_Group_t

QAPI console command group.

3.1.3 Function Documentation

3.1.3.1 QAPI_Console_Group_Handle_t QAPI_Console_Register_Command_Group (QAPI_Console_Group_Handle_t *Parent_Group,* const QAPI_Console_Command_Group_t * *Command_Group*)

Registers a command group with the CLI.

Parameters

Parent_Group	Group to which this group should be registerd under as a subgroup. If this parameter is NULL, then the group will be registered at the top level.
Command_Group	Command group to be registered. This function assumes that the command group information will be constant and simply stores a pointer to the data. If the command group and its associated information is not constant, its memory must be retained until the command is unregistered.

Returns

- Handle for the group that was added.
- NULL if there was an error registering the group.

3.1.3.2 QAPI_Console_Handle_t QAPI_Console_Register_Command (QAPI_Console-_Group_Handle_t *Parent_Group,* const QAPI_Console_Command_t * *Command*)

Registers a command with the CLI.

Parameters

Parent_Group	Group to which this group should be registerd under as a subgroup. If this parameter is NULL, then the command will be registered at the top level.
Command	Command to be registered.

Returns

- Handle for the command that was added.
- NULL if there was an error registering the command.

3.2 Console firmware upgrade codes

The following definitions represent the status codes of the Firmware Upgrade module.

3.2.1 Define Documentation

3.2.1.1 #define QAPI_ERROR_CONSOLE_COMMAND_STATUS_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_CONSOLE, 1)))

Console command generic error.

3.2.1.2 #define QAPI_ERROR_CONSOLE_COMMAND_STATUS_USAGE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_CONSOLE, 2)))

Return usage for console command.

3.2.1.3 #define QAPI_ERROR_CONSOLE_INIT_FAIL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_CONSOLE, 3)))

Failure to initialize console.

3.2.1.4 #define QAPI_ERROR_CONSOLE_REGISTER_CMD_GROUP_FAIL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_CONSOLE, 4)))

Failure to register command group.

3.2.1.5 #define QAPI_ERROR_CONSOLE_REGISTER_CMD_FAIL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_CONSOLE, 5)))

Failure to register command.

4 Firmware Upgrade module

4.1 Firmware upgrade APIs

4.1.1 Data Structure Documentation

4.1.1.1 struct qapi_Fw_Upgrade_Plugin_t

Represents a set of firmware upgrade plugin callbacks.

When the application calls qpai_Fw_Upgrade(), it must fill this structure and pass it to the firmware upgrade engine. The engine calls these firmware upgrade plugin callbacks during different stages of an upgrade.

Data fields

Type	Parameter	Description
qapi_Fw_-Upgrade_-Plugin_Init_t	fw_Upgrade_-Plugin_Init	Callback to initialize a firmware upgrade.
qapi_Fw_-Upgrade_-Plugin_Recv_-Data_t	fw_Upgrade_-Plugin_Recv_-Data	Callback to retrieve data.
qapi_Fw_-Upgrade_-Plugin_Abort_t	fw_Upgrade_-Plugin_Abort	Firmware upgrade plugin abort callback.
qapi_Fw-_Upgrade-_Plugin_-Resume_t	fw_Upgrade_-Plugin_Resume	Firmware upgrade plugin resume callback.
qapi_Fw_-Upgrade_-Plugin_Fin_t	fw_Upgrade_-Plugin_Fin	Firmware upgrade plugin finish callback.

4.1.2 Typedef Documentation

4.1.2.1 typedef void* qapi_Part_Hdl_t

Defines an opaque Firmware Partition Handle type.

A Partition Handle refers to some partition in a valid or semi-valid (under construction) Firmware Descriptor.

4.1.2.2 typedef void(∗ qapi_Fw_Upgrade_CB_t)(int32_t state, int32_t status)

Declaration of a callback function called by the firmware upgrade state machine. The application implements this callback and passes it as a parameter to the qapi_Fw_Upgrade() API.

Parameters

in	*state*	Firmware upgrade state machine state defined by enum #qapi_Fw_Upgrade_State_t.
in	*status*	QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

Returns

None.

4.1.2.3 typedef qapi_Status_t(∗ qapi_Fw_Upgrade_Plugin_Init_t)(const char ∗interface_Name, const char ∗url, void ∗init_Param)

Declaration of a callback function called by the firmware upgrade state machine on initialization. The plugin module implements this callback and performs all plugin related initializations. The application passes this callback as a parameter to the qapi_Fw_Upgrade() API.

Parameters

in	*interface_Name*	Network interface name (plugin dependent).
in	*url*	Server URL (plugin dependent).
in	*init_Param*	Initialization parameters (plugin dependent).

Returns

Status QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

4.1.2.4 typedef qapi_Status_t(∗ qapi_Fw_Upgrade_Plugin_Fin_t)(void)

Declaration of a callback function called by the firmware upgrade state machine on upgrade completion. The plugin module implements this callback and performs all plugin related cleanup. The application passes this callback as a parameter to the qapi_Fw_Upgrade() API.

Returns

Status QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

4.1.2.5 typedef qapi_Status_t(∗ qapi_Fw_Upgrade_Plugin_Recv_Data_t)(uint8_t ∗buffer, uint32_t buf_len, uint32_t ∗ret_size, void ∗init_Param)

Declaration of a callback function called by the firmware upgrade state machine to receieve a packet from the plugin. The plugin module implements this callback and fills the buffer with incoming data. The application passes this callback as a parameter to the qapi_Fw_Upgrade() API.

Parameters

out	*buffer*	Receive data buffer.
in	*buf_len*	Buffer length.
out	*ret_size*	Received data size.
in	*init_Param*	Optional initialization parameters.

Returns

Status QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

4.1.2.6 typedef qapi_Status_t(∗ qapi_Fw_Upgrade_Plugin_Abort_t)(void)

Declaration of a callback function called by the firmware upgrade state machine to abort a plugin operation. The plugin module implements this callback and aborts connection when invoked. The application passes this callback as a parameter to the qapi_Fw_Upgrade() API.

Returns

Status QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

4.1.2.7 typedef qapi_Status_t(∗ qapi_Fw_Upgrade_Plugin_Resume_t)(const char ∗interface_name, const char ∗url, const uint32_t offset)

Declaration of a callback function called by the firmware upgrade state machine on resume. The plugin module implements this callback and performs all plugin related resumes. The application passes this callback as a parameter to the qapi_Fw_Upgrade() API.

Parameters

in	*interface_name*	Network interface name (plugin dependent).
in	*url*	Server URL (plugin dependent).
in	*offset*	Offset to resume the download (plugin dependent).

Returns

Status QAPI_OK or error code #QAPI_FW_UPGRADE_ERR_XXX.

4.1.3 Enumeration Type Documentation

4.1.3.1 enum qapi_Fw_Upgrade_State

Enumeration that represents the various states in Firmware Upgrade state machine.

Enumerator:

QAPI_FW_UPGRADE_STATE_NOT_START_E Firmware upgrade operation is not started.
QAPI_FW_UPGRADE_STATE_GET_TRIAL_INFO_E Get trial image information at flash.
QAPI_FW_UPGRADE_STATE_ERASE_FWD_E Erase FWD.
QAPI_FW_UPGRADE_STATE_ERASE_FLASH_E Erase the partition.
QAPI_FW_UPGRADE_STATE_ERASE_SECOND_FS_E Erase the second file system.
QAPI_FW_UPGRADE_STATE_PREPARE_FS_E Prepare the file system.

QAPI_FW_UPGRADE_STATE_ERASE_IMAGE_E Erase the subimage.
QAPI_FW_UPGRADE_STATE_PREPARE_CONNECT_E Prepare to connect to a remote firmware upgrade server.
QAPI_FW_UPGRADE_STATE_CONNECT_SERVER_E Connect to a remote firmware upgrade server.
QAPI_FW_UPGRADE_STATE_RESUME_SERVICE_E Resume the firmware upgrade service.
QAPI_FW_UPGRADE_STATE_RESUME_SERVER_E Resume connecting to the firmware upgrade server.
QAPI_FW_UPGRADE_STATE_RECEIVE_DATA_E Receive data from the remote firmware upgrade server.
QAPI_FW_UPGRADE_STATE_DISCONNECT_SERVER_E Disconnected from a remote firmware upgrade server.
QAPI_FW_UPGRADE_STATE_PROCESS_CONFIG_FILE_E Process firmware upgrade configuration file.
QAPI_FW_UPGRADE_STATE_PROCESS_IMAGE_E Process the image.
QAPI_FW_UPGRADE_STATE_DUPLICATE_IMAGES_E Duplicate the images from the current FWD.
QAPI_FW_UPGRADE_STATE_DUPLICATE_FS_E Duplicate the file system.
QAPI_FW_UPGRADE_STATE_FINISH_E Firmware upgrade is done.

4.1.4 Function Documentation

4.1.4.1 qapi_Status_t qapi_Fw_Upgrade (char * *interface_Name*, qapi_Fw_Upgrade_Plugin_t * *plugin*, char * *url*, char * *cfg_File*, uint32_t *flags*, qapi_Fw_Upgrade_CB_t *cb*, void * *init_Param*)

Starts a firmware upgrade session.

The caller from the application domain specifies all the required parameters, including the plugin functions, source of the image, and flags. The session automatically ends in the case of an error.

Parameters

in	*interface_Name*	Network interface name, e.g., wlan1.
in	*plugin*	Parameter of type qapi_Fw_Upgrade_Plugin_t containing a set of plugin callback functions. For more details, refer to qapi_Fw_Upgrade_Plugin_t.
in	*url*	Source information for a firmware upgrade. For FTP: [user_name]:[password]@[IPV4 address]:[port] for IPV4 [user_name]:[password]@|[IPV6 address]|:[port] for IPV6
in	*cfg_File*	Image file information for a firmware upgrade.
in	*flags*	Flags with bits defined for a firmware upgrade. See the qapi_Fw_Upgrade flag for a definition.
in	*cb*	Optional callback function called by firmware upgrade engine to provide status information.
in	*init_Param*	Optional init parameter passed to the firmware upgrade init function.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.2 qapi_Status_t qapi_Fw_Upgrade_Cancel (void)

Cancels a firmware upgrade session.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.3 qapi_Status_t qapi_Fw_Upgrade_Suspend (void)

Suspends the firmware upgrade session.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.4 qapi_Status_t qapi_Fw_Upgrade_Resume (void)

Resumes the firmware upgrade session.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.5 qapi_Status_t qapi_Fw_Upgrade_Done (uint32_t *result,* uint32_t *flags*)

Activates or invalidates the trial image. The application calls this API after it has verified that the image is valid or invalid. The criteria for image validity is defined by the application.

Parameters

in	*result*	Result: 1: The image is valid; set trial image to active 0: The image is invalid; invalidate trial image.
in	*flags*	Flags (bit0): 1: The device reboots after activating or invalidating the trial image 0: The device does not reboot after activating or invalidating the trial image

Note

If the reboot_flag is set, the device will reboot and there is no return.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.6 qapi_Status_t qapi_Fw_Upgrade_init (void)

Initializes Firmware Upgrade library.

Reads Firmware Descriptors and populates internal data structures. Must be called before other firmware descriptor or partition related APIs are used.

Returns

On success, QAPI_OK is returned; on error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.7 qapi_Status_t qapi_Fw_Upgrade_Erase_FWD (uint8_t *FWD_idx*)

Erases a firmware descriptor so that an entirely new FWD can be formed in its place.

Parameters

in	*FWD_idx*	Firmware descriptor index number to be erased.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.8 qapi_Status_t qapi_Fw_Upgrade_Get_FWD_Magic (uint8_t *FWD_idx,* uint32_t * *magic*)

Gets the magic number from a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
out	*magic*	Magic number of the firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.9 qapi_Status_t qapi_Fw_Upgrade_Set_FWD_Magic (uint8_t *FWD_idx,* uint32_t *magic*)

Sets the magic number for a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
in	*magic*	Magic number to write to the firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.10 qapi_Status_t qapi_Fw_Upgrade_Get_FWD_Rank (uint8_t *FWD_idx,* uint32_t * *rank*)

Gets the rank number from a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
out	*rank*	Rank number at the firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.11 qapi_Status_t qapi_Fw_Upgrade_Get_FWD_Version (uint8_t *FWD_idx,* uint32_t * *version*)

Gets the version number from a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
out	*version*	Version number of the firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.12 qapi_Status_t qapi_Fw_Upgrade_Get_FWD_Status (uint8_t *FWD_idx,* uint8_t * *status*)

Gets the status value from a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
out	*status*	Status value of the firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.13 qapi_Status_t qapi_Fw_Upgrade_Get_FWD_Total_Images (uint8_t *FWD_idx,* uint8_t * *image_nums*)

Gets the total number of images for a firmware descriptor.

Parameters

in	*FWD_idx*	Firmware descriptor index number to operate.
out		Image numbers from firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.14 uint8_t qapi_Fw_Upgrade_Get_Active_FWD (uint32_t * *fwd_boot_type,* uint32_t * *valid_fwd*)

Gets the FWD index number that is current running.

This is for the FWD that was selected by the bootloaders and is currently in use.

Parameters

out	*fwd_boot_type*	Type of FWD used for booting.
out	*valid_fwd*	Information about which FWDs are present (1 bit per FWD).

Returns

The active FWD number.

4.1.4.15 qapi_Status_t qapi_Fw_Upgrade_Close_Partition (qapi_Part_Hdl_t *hdl*)

Releases a partition handle.

Parameters

in	*hdl*	partition handle

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.16 qapi_Status_t qapi_Fw_Upgrade_First_Partition (uint8_t *FWD_idx,* qapi_Part_Hdl_t * *hdl*)

Gets a handle for the first partition associated with the specified FWD.

Parameters

in	*FWD_idx*	Firmware descriptor index number.
out	*hdl*	Partition handle for the partition operation.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.17 qapi_Status_t qapi_Fw_Upgrade_Next_Partition (qapi_Part_Hdl_t *curr,* qapi_Part_Hdl_t * *hdl*)

Gets the next partition after the current one.

Guaranteed to be in the same FWD as curr. This function returns an error when it reaches all blank (uninitialized) partition metadata.

Parameters

in	*curr*	Current partition handle.
out	*hdl*	Next partition handle.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.18 qapi_Status_t qapi_Fw_Upgrade_Find_Partition (uint8_t *FWD_idx,* uint32_t *id,* qapi_Part_Hdl_t * *hdl*)

Finds a partition.

This function scans the partition table associated with the specified FWD to find the first partition with the specified ID and get a handle to that partition.

Parameters

in	*FWD_idx*	Firmware descriptor index number.
in	*id*	Partition image ID.
out	*hdl*	Partition handle.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.19 qapi_Status_t qapi_Fw_Upgrade_Get_Image_ID (qapi_Part_Hdl_t *hdl*, uint32_t * *id*)

Gets the image ID associated with a partition in a FWD.

Parameters

in	*hdl*	Partition handle.
out	*id*	Partition image ID.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.20 qapi_Status_t qapi_Fw_Upgrade_Get_Image_Version (qapi_Part_Hdl_t *hdl*, uint32_t * *version*)

Gets the image version associated with a partition in a FWD.

Parameters

in	*hdl*	Partition handle.
out	*version*	Image version to retrieve.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.21 qapi_Status_t qapi_Fw_Upgrade_Get_Partition_Size (qapi_Part_Hdl_t *hdl*, uint32_t * *size*)

Gets the size of a partition in a FWD.

Parameters

in	*hdl*	Partition handle.
out	*size*	Partition image size.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.22 qapi_Status_t qapi_Fw_Upgrade_Set_Image_Size (qapi_Part_Hdl_t * *hdl*, uint32_t *size*)

Sets size of the associated partition in a FWD to zero (0).

Parameters

in	*hdl*	Partition handle. Cannot be the handle of APP or SBL image.
in	*size*	Image size to be set. Currently, size can only be 0; otherwise, function returns error code.

Detailed description

Setting the size of an APP or SBL image is not allowed. Set size to non-zero value is not allowed. Calling this function repeatedly will cause frequent flash writing, which reduces the life of flash (not recommended).

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.23 qapi_Status_t qapi_Fw_Upgrade_Get_Partition_Start (qapi_Part_Hdl_t *hdl,* uint32_t * ***start***)

Gets the start offset of a partition in a FWD.

Parameters

in	*hdl*	Partition handle.
out	*start*	Start offset.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.24 qapi_Status_t qapi_Fw_Upgrade_Get_Partition_FWD (qapi_Part_Hdl_t *hdl,* uint8_t * ***FWD_idx***)

Identifies with which FWD a partition handle is associated.

Parameters

in	*hdl*	Partition handle.
out	*FWD_idx*	Firmware descriptor index number.

On success, QAPI_OK is returned.

On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.25 qapi_Status_t qapi_Fw_Upgrade_Erase_Partition (qapi_Part_Hdl_t *hdl*, uint32_t *offset*, uint32_t *nbytes*)

Erases n bytes of memory starting at the specified offset from the start of the specified partition.

Offset and nbytes must be multiples of FLASH_BLOCK_SZ (4 KB) if it is flash area. This is a partition-relative byte-oriented erase operation.

Parameters

in	*hdl*	Partition handle.
in	*offset*	Memory offset to erase.
in	*nbytes*	Memory size to erase.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.26 qapi_Status_t qapi_Fw_Upgrade_Write_Partition (qapi_Part_Hdl_t *hdl*, uint32_t *offset*, char * *buf*, uint32_t *nbytes*)

Writes n bytes from a specified buffer to memory at the specified offset from the start of the specified partition.

This is a partition-relative byte-oriented write operation. Either the partition must be blank (erased) before this write operation begins or the write operation must avoid any attempt to change a 0 to a 1 bit if it is flash. If the final contents do not match the original buffer, an error is raised.

Parameters

in	*hdl*	Partition handle.
in	*offset*	Memory offset to write.
in	*buf*	Buffer point to write to memory.
in	*nbytes*	Memory size to write.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.1.4.27 qapi_Status_t qapi_Fw_Upgrade_Read_Partition (qapi_Part_Hdl_t *hdl*, uint32_t *offset*, char * *buf*, uint32_t *max_bytes*, uint32_t * *nbytes*)

Reads up to max_bytes into the specified buffer from memory at the specified offset from the start of the specified partition.

This is a partition-relative byte-oriented read operation.

Parameters

in	*hdl*	Partition handle.
in	*offset*	Memory offset to read.
in	*buf*	Buffer point to store the memory data.
in	*max_bytes*	Size to read.
out	*nbytes*	Actual memory read size in buf.

Returns

On success, QAPI_OK is returned.
On error, error code #QAPI_FW_UPGRADE_ERR_XXX is returned.

4.2 FW upgrade definitions

4.2.1 Define Documentation

4.2.1.1 #define QAPI_FW_UPGRADE_FLAG_AUTO_REBOOT (1<<0)

Definition used by the qapi_Fw_Upgrade() and qapi_Fw_Upgrade_done() APIs as a flag bit. The Fw_Upgrade and Fw_Upgrade_Done APIs automatically reboot if this flag bit is set.

4.2.1.2 #define QAPI_FW_UPGRADE_FLAG_DUPLICATE_ACTIVE_FS (1<<1)

Definition used by the qapi_Fw_Upgrade() API as a flag. Fw_Upgrade copies files from an active image file system to a trial image file system if this flag is set

4.2.1.3 #define QAPI_FW_UPGRADE_FLAG_RANGE_HEADER (1<<2)

Definition used by the qapi_Fw_Upgrade() API as a flag. Fw_Upgrade get active image file with range header as a flag for ota_httpc_get_with_offset if this flag is set.

4.3 FWD bit information

Definition used by the qapi_Fw_Upgrade_Get_Active_FWD() API as a return to indicate the FWD bit type

4.3.1 Define Documentation

4.3.1.1 #define QAPI_FW_UPGRADE_FWD_BIT_GOLDEN (0)

4.3.1.2 #define QAPI_FW_UPGRADE_FWD_BIT_CURRENT (1)

4.3.1.3 #define QAPI_FW_UPGRADE_FWD_BIT_TRIAL (2)

4.4 FWD boot type information

Definition used by the qapi_Fw_Upgrade_Get_Active_FWD() API as a return to indicate the FWD type for booting.

4.4.1 Define Documentation

4.4.1.1 #define QAPI_FW_UPGRADE_FWD_BOOT_TYPE_GOLDEN (1<<QAPI_FW_UPGRADE_FWD_BIT_GOLDEN)

4.4.1.2 #define QAPI_FW_UPGRADE_FWD_BOOT_TYPE_CURRENT (1<<QAPI_FW_UPGRADE_FWD_BIT_CURRENT)

4.4.1.3 #define QAPI_FW_UPGRADE_FWD_BOOT_TYPE_TRIAL (1<<QAPI_FW_UPGRADE_FWD_BIT_TRIAL)

4.5 FWUP return status information

Definitions used by the firmware upgrade API as a return to indicate the status.

4.5.1 Define Documentation

4.5.1.1 #define QAPI_FW_UPGRADE_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 1)

< Operation failed. Invalid parameter.

4.5.1.2 #define QAPI_FW_UPGRADE_ERR_INVALID_PARAM __QAPI_ERROR(QAPI_MOD_FWUP, 2)

Firmware upgrade library is not initialized.

4.5.1.3 #define QAPI_FW_UPGRADE_ERR_NOT_INIT __QAPI_ERROR(QAPI_MOD_FWUP, 3)

Operation is incomplete.

4.5.1.4 #define QAPI_FW_UPGRADE_ERR_INCOMPLETE __QAPI_ERROR(QAPI_MOD_FWUP, 4)

Firmware upgrade session is inprogress.

4.5.1.5 #define QAPI_FW_UPGRADE_ERR_SESSION_IN_PROGRESS __QAPI_ERROR(QAPI_MOD_FWUP, 5)

Firmware upgrade session is not started.

4.5.1.6 #define QAPI_FW_UPGRADE_ERR_SESSION_NOT_START __QAPI_ERROR(QAPI_MOD_FWUP, 6)

Firmware upgrade session is not ready to enter the Suspend state.

4.5.1.7 #define QAPI_FW_UPGRADE_ERR_SESSION_NOT_READY_FOR_SUSPEND __QAPI_ERROR(QAPI_MOD_FWUP, 7)

Firmware upgrade session is not in the Suspend state.

4.5.1.8 #define QAPI_FW_UPGRADE_ERR_SESSION_NOT_SUSPEND __QAPI_ERROR(QAPI_MOD_FWUP, 8)

Firmware upgrade session resume is not supported by the plugin.

4.5.1.9 #define QAPI_FW_UPGRADE_ERR_SESSION_RESUME_NOT_SUPPORT __QAPI_ERROR(QAPI_MOD_FWUP, 9)

Firmware upgrade session was cancelled.

4.5.1.10 #define QAPI_FW_UPGRADE_ERR_SESSION_CANCELLED __QAPI_ERROR(QAPI_MOD_FWUP, 10)

Firmware upgrade session was suspended.

4.5.1.11 #define QAPI_FW_UPGRADE_ERR_SESSION_SUSPEND __QAPI_ERROR(QAPI_MOD_FWUP, 11)

Interface name is too long.

4.5.1.12 #define QAPI_FW_UPGRADE_ERR_INTERFACE_NAME_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 12)

URL is too long.

4.5.1.13 #define QAPI_FW_UPGRADE_ERR_URL_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 13)

Not supported firmware upgrade.

4.5.1.14 #define QAPI_FW_UPGRADE_ERR_FLASH_NOT_SUPPORT_FW_UPGRADE - __QAPI_ERROR(QAPI_MOD_FWUP, 14)

Flash initialization timeout.

4.5.1.15 #define QAPI_FW_UPGRADE_ERR_FLASH_INIT_TIMEOUT __QAPI_ERROR(-QAPI_MOD_FWUP, 15)

Flash read failure.

4.5.1.16 #define QAPI_FW_UPGRADE_ERR_FLASH_READ_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 16)

Flash write failure.

4.5.1.17 #define QAPI_FW_UPGRADE_ERR_FLASH_WRITE_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 17)

Flash erase failure.

4.5.1.18 #define QAPI_FW_UPGRADE_ERR_FLASH_ERASE_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 18)

Not enough free space in flash.

4.5.1.19 #define QAPI_FW_UPGRADE_ERR_FLASH_NOT_ENOUGH_SPACE __QAPI_ERROR(QAPI_MOD_FWUP, 19)

Partition creation failure.

4.5.1.20 #define QAPI_FW_UPGRADE_ERR_FLASH_CREATE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 20)

Partition image was not found.

4.5.1.21 #define QAPI_FW_UPGRADE_ERR_FLASH_IMAGE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 21)

Partition erase failure.

4.5.1.22 #define QAPI_FW_UPGRADE_ERR_FLASH_ERASE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 22)

Partition write failure.

4.5.1.23 #define QAPI_FW_UPGRADE_ERR_FLASH_WRITE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 23)

NULL partition.

4.5.1.24 #define QAPI_FW_UPGRADE_ERR_GET_PARTITION_NULL __QAPI_ERROR(QAPI_MOD_FWUP, 24)

Reach max image entry.

4.5.1.25 #define QAPI_FW_UPGRADE_ERR_REACH_MAX_IMAGE_ENTRY __QAPI_ERROR(QAPI_MOD_FWUP, 25)

Image Entry is not used

4.5.1.26 #define QAPI_FW_UPGRADE_ERR_IMAGE_UNUSED __QAPI_ERROR(QAPI_MOD_FWUP, 26)

Image not found failure.

4.5.1.27 #define QAPI_FW_UPGRADE_ERR_IMAGE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 27)

Image download failure.

4.5.1.28 #define QAPI_FW_UPGRADE_ERR_IMAGE_DOWNLOAD_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 28)

Incorrect image checksum failure.

4.5.1.29 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_CHECKSUM __QAPI_ERROR(QAPI_MOD_FWUP, 29)

Server communication timeout.

4.5.1.30 #define QAPI_FW_UPGRADE_ERR_SERVER_RSP_TIMEOUT __QAPI_ERROR(QAPI_MOD_FWUP, 30)

Image file name is invalid.

4.5.1.31 #define QAPI_FW_UPGRADE_ERR_INVALID_FILENAME __QAPI_ERROR(QAPI_MOD_FWUP, 31)

Firmware upgrade image header is invalid.

4.5.1.32 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_HDR __QAPI_ERROR(QAPI_MOD_FWUP, 32)

Not enough memory.

4.5.1.33 #define QAPI_FW_UPGRADE_ERR_INSUFFICIENT_MEMORY __QAPI_ERROR(QAPI_MOD_FWUP, 33)

Firmware upgrade image signature is invalid.

4.5.1.34 #define QAPI_FW_UPGRADE_ERR_INCORRECT_SIGNATURE __QAPI_ERROR(QAPI_MOD_FWUP, 34)

Firmware upgrade image version is invalid.

4.5.1.35 #define QAPI_FW_UPGRADE_ERR_INCORRCT_VERSION __QAPI_ERROR(QAPI_MOD_FWUP, 35)

Firmware upgrade image number of images is invalid.

4.5.1.36 #define QAPI_FW_UPGRADE_ERR_INCORRECT_NUM_IMAGES __QAPI_ERROR(QAPI_MOD_FWUP, 36)

Firmware upgrade image length is invalid.

4.5.1.37 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_LENGTH __QAPI_ERROR(QAPI_MOD_FWUP, 37)

Firmware upgrade image hash type is invalid.

4.5.1.38 #define QAPI_FW_UPGRADE_ERR_INCORRECT_HASH_TYPE __QAPI_ERROR(QAPI_MOD_FWUP, 38)

Firmware upgrade image ID is invalid.

4.5.1.39 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_ID __QAPI_ERROR(QAPI_MOD_FWUP, 39)

SBL upgrade only is not supported.

4.5.1.40 #define QAPI_FW_UPGRADE_ERR_SBL_ONLY_NOT_SUPPORT __QAPI_ERROR(QAPI_MOD_FWUP, 40)

SBL upgrade not supported.

4.5.1.41 #define QAPI_FW_UPGRADE_ERR_SBL_NOT_SUPPORT_UPGRADE __QAPI_ERROR(QAPI_MOD_FWUP, 41)

No enough memory for SBL upgrade.

4.5.1.42 #define QAPI_FW_UPGRADE_ERR_SBL_NOT_ENOUGH_SPACE __QAPI_ERROR(QAPI_MOD_FWUP, 42)

Invalid FDT.

4.5.1.43 #define QAPI_FW_UPGRADE_ERR_INVALID_FDT __QAPI_ERROR(QAPI_MOD_FWUP, 43)

Battery level is too low.

4.5.1.44 #define QAPI_FW_UPGRADE_ERR_BATTERY_LEVEL_TOO_LOW __QAPI_ERROR(QAPI_MOD_FWUP, 44)

Crypto check failure.

4.5.1.45 #define QAPI_FW_UPGRADE_ERR_CRYPTO_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 45)

Firmware upgrade plugin callback is empty.

4.5.1.46 #define QAPI_FW_UPGRADE_ERR_PLUGIN_ENTRY_EMPTY __QAPI_ERROR(QAPI_MOD_FWUP, 46)

Trial image is running

4.5.1.47 #define QAPI_FW_UPGRADE_ERR_TRIAL_IS_RUNNING __QAPI_ERROR(QAPI_MOD_FWUP, 47)

File was not found.

4.5.1.48 #define QAPI_FW_UPGRADE_ERR_FILE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 48)

Open file failure.

4.5.1.49 #define QAPI_FW_UPGRADE_ERR_FILE_OPEN_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 49)

File name is too long.

4.5.1.50 #define QAPI_FW_UPGRADE_ERR_FILE_NAME_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 50)

Write file failure.

4.5.1.51 #define QAPI_FW_UPGRADE_ERR_FILE_WRITE_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 51)

Mount file system failure.

4.5.1.52 #define QAPI_FW_UPGRADE_ERR_MOUNT_FILE_SYSTEM_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 52)

Firmware upgrade create thread failure.

5 Flash module

The QSPI Flash module provides flash operation APIs.

qapi_Flash_Init() initializes the flash module and must be called before any other function in this module. The system automatically searches for the connected flash.

If need to change some flash configuration, refer to flash_config_data_t flash_device_config[].

Flash read/write/erase/readreg/writereg operations cannot be performed simultaneously. There can be only one operation ongoing at a time.

Flash read/write/erase/readreg/writereg support only blocking operations.

For a blocking operation, the related function returns until the required number of data/status are read/written/erased successfully, or there is an error. If the required number is large, the related function may take some time to return. It should be taken into consideration if the product is power and time sensitive.

5.1 Flash APIs

The QSPI Flash module provides flash operation APIs.

qapi_Flash_Init() initializes the flash module and must be called before any other function in this module. The system automatically searches for the connected flash.

If need to change some flash configuration, refer to flash_config_data_t flash_device_config[].

Flash read/write/erase/readreg/writereg operations cannot be performed simultaneously. There can be only one operation ongoing at a time.

Flash read/write/erase/readreg/writereg support only blocking operations.

For a blocking operation, the related function returns until the required number of data/status are read/written/erased successfully, or there is an error. If the required number is large, the related function may take some time to return. It should be taken into consideration if the product is power and time sensitive.

5.1.1 Define Documentation

5.1.1.1 #define QAPI_FLASH_DEVICE_FAIL __QAPI_ERROR(QAPI_MOD_FLASH, 1)

Operation failed

5.1.1.2 #define QAPI_FLASH_DEVICE_NOT_SUPPORTED __QAPI_ERROR(QAPI_MOD_FLASH, 2)

Device/operation not supported

5.1.1.3 #define QAPI_FLASH_DEVICE_INVALID_PARAMETER __QAPI_ERROR(QAPI_MOD_FLASH, 3)

API parameters invalid

5.1.1.4 #define QAPI_FLASH_DEVICE_IMAGE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FLASH, 4)

FW Image ID not found

5.1.1.5 #define QAPI_FLASH_DEVICE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FLASH, 5)

Device not found on supported device list

5.1.1.6 #define QAPI_FLASH_DEVICE_PENDING __QAPI_ERROR(QAPI_MOD_FLASH, 6)

Flash non-blocking operation is ongoing.

5.1.1.7 #define QAPI_FLASH_DEVICE_NO_MEMORY __QAPI_ERROR(QAPI_MOD_FLASH, 7)

Memory allocation failed.

5.1.1.8 #define QAPI_FLASH_DEVICE_BUSY __QAPI_ERROR(QAPI_MOD_FLASH, 8)

Flash device is busy.

5.1.2 Data Structure Documentation

5.1.2.1 struct flash_info_s

Flash client device data, should be same with drv_flash_info_t

Data fields

Type	Parameter	Description
uint32_t	devicd_id	Capacity ID + Memory Type ID + Manufacturer ID.
uint32_t	block_count	Number of total blocks for this partition/image.
uint16_t	block_size_bytes	Block size in bytes.
uint16_t	page_size_bytes	Page size in bytes.

5.1.3 Typedef Documentation

5.1.3.1 typedef struct flash_info_s flash_info_t

Flash client device data, should be same with drv_flash_info_t

5.1.4 Enumeration Type Documentation

5.1.4.1 enum qapi_FLASH_Erase_Type_t

Enumeration of flash erase types.

Enumerator:

QAPI_FLASH_BLOCK_ERASE_E Block erase. Block size is 4KB.

QAPI_FLASH_BULK_ERASE_E Bulk erase. Bulk size is n∗4Kb, for example 32Kb, 64Kb, and so on. It depends on the specific flash type. Here is 64Kb by default

QAPI_FLASH_CHIP_ERASE_E Chip erase.

5.1.5 Function Documentation

5.1.5.1 qapi_Status_t qapi_Flash_Init ( )

Initialize the flash module.

This function must be called before any other flash functions.

Returns

QAPI_OK – On success.
Error code – On failure.

5.1.5.2 qapi_Status_t qapi_Flash_Read (uint32_t *Address,* uint32_t *ByteCnt,* uint8_t ∗ *Buffer*)

Read data from the flash.

Parameters

in	*Address*	The flash address to start to read from.
in	*ByteCnt*	Number of bytes to read.
out	*Buffer*	Data buffer for a flash read operation.

Returns

QAPI_OK – If a read completed successfully.
Error code – If there is an error.

5.1.5.3 qapi_Status_t qapi_Flash_Write (uint32_t *Address,* uint32_t *ByteCnt,* uint8_t ∗ *Buffer*)

Write data to the flash.

Parameters

in	*Address*	The flash address to start to write to.
in	*ByteCnt*	Number of bytes to write.
in	*Buffer*	Data buffer containing data to be written.

Returns

QAPI_OK – If blocking write completed successfully.
Error code – If there is an error.

5.1.5.4 qapi_Status_t qapi_Flash_Erase (qapi_FLASH_Erase_Type_t *EraseType,* uint32_t *Start,* uint32_t *Cnt*)

Erase the given flash blocks or bulks, or the whole chip.

Parameters

in	*EraseType*	Specify the erase type.
in	*Start*	For block erase - the starting block of a number of blocks to erase. For bulk erase - the starting bulk of a number of bulks to erase. For chip erase, it should be 0.
in	*Cnt*	For block erase - the number of blocks to erase. For bulk erase - the number of bulks to erase. For chip erase, it should be 1.

Returns

QAPI_OK – If blocking erase completed successfully.
Error code – If there was an error.

5.1.5.5 qapi_Status_t qapi_Flash_Get_Info (flash_info_t * *flash_info*)

Get flash device info in use.

Parameters

in	*RegOpcode*	Operation code.
in	*Len*	The length of register value to be read.
out	*RegValue*	The read out value.

Returns

QAPI_OK – On success.
Error code – On failure.

6 Heap Status module

6.1 Heap status APIs

6.1.1 Data Structure Documentation

6.1.1.1 struct heap_status_t

Heap status.

Data fields

Type	Parameter	Description
uint32_t	total_Bytes	
uint32_t	free_Bytes	
uint32_t	min_ever_free-_bytes	
uint32_t	lwip_total_-Bytes	
uint32_t	lwip_free_-Bytes	
uint32_t	lwip_min_ever-_free_bytes	
uint32_t	lwip_total_pool	
uint32_t	lwip_free_pool	
uint32_t	lwip_min_ever-_free_pool	

6.1.2 Function Documentation

6.1.2.1 qapi_Status_t qapi_Heap_Status (heap_status * *hs*)

Gets the memory heap status, including total bytes and free bytes.

Parameters

`out`	*hs*	Pointer to hold the heap status structure

Returns

QAPI_OK on success, or a different code on error.

7 I2C module

The I²C module uses an inter-integrated circuit interface (I²C) 2-wire bus to connect low-speed peripherals to a processor or a microcontroller. Common I²C peripherals include touch screen controllers, accelerometers, gyros, ambient light, and temperature sensors.

The 2-wire bus comprises a data line, a clock line, and basic START, STOP, and acknowledge signals to drive transfers on the bus. An I²C peripheral is also referred to as an I²C slave. The processor or microcontroller implements the I²C master as defined in the I²C specification. This documentation provides the software interface to access the I²C master implementation.

NOTE Before using module APIs, qapi_PWR_Set_Module_State (#QAPI_PWR_MODULE_PERIPHERAL_E, #QAPI_PWR_STATE_ACTIVE_E, #QAPI_PWR_STATE_ACTIVE_E) must be called.

7.1 I2C APIs

7.1.1 Data Structure Documentation

7.1.1.1 struct qapi_I2CM_Config_s

I²C master configuration parameters.

Data fields

Type	Parameter	Description
qbool_t	Blocking	**Supported values:** • 1 – Blocking mode • 0 – Nonblocking mode.
qbool_t	Dma	**Supported values:** • TRUE – DMA • FALSE – FIFO

7.1.1.2 struct qapi_I2CM_Transfer_Config_s

I²C master client configuration parameters that the client uses to communicate to an I²C slave.

Data fields

Type	Parameter	Description
uint32_t	BusFreqKHz	I²C master bus speed in kHz.
uint32_t	SlaveAddress	7-bit I²C slave address.
uint32_t	SlaveMax-ClockStretchUs	Maximum slave clock stretch in µs.
uint32_t	Delay	Delay before the start and at the end of a command. Recommended 0.
uint32_t	NoiseReject	Noise reject. Recommended 0.

7.1.1.3 struct qapi_I2CM_Descriptor_s

I²C master transfer descriptor.

Data fields

Type	Parameter	Description
uint8_t *	Buffer	Buffer for the data transfer.
uint32_t	Length	Length of the data to be transferred in bytes.
uint32_t	Transferred	Number of bytes transferred.
uint32_t	Flags	I²C master flags for the transfer.

7.1.2 Typedef Documentation

7.1.2.1 typedef struct qapi_I2CM_Config_s qapi_I2CM_Config_t

I²C master configuration parameters.

7.1.2.2 typedef struct qapi_I2CM_Transfer_Config_s qapi_I2CM_Transfer_Config_t

I²C master client configuration parameters that the client uses to communicate to an I²C slave.

7.1.2.3 typedef struct qapi_I2CM_Descriptor_s qapi_I2CM_Descriptor_t

I²C master transfer descriptor.

7.1.2.4 typedef void(∗ qapi_I2CM_Transfer_CB_t)(uint32_t Status, void ∗CallbackCtxt)

Prototype for a function called when the data is completely transferred on the bus, or when the transfer ends due to an error or cancellation. Clients pass the callback function pointer and the callback context to the driver in the qapi_I2CM_Transfer() API.

Note

The callback is called in the I²C master driver context. Calling any of the I²C master QAPIs defined in this file in the callback is forbidden.

Parameters

out	*Status*	The completion status of the transfer.
out	*CallbackCtxt*	The callback context that was passed in the call to qapi_I2CM_Transfer().

7.1.3 Enumeration Type Documentation

7.1.3.1 enum qapi_I2CM_Instance_t

Instance of the I²C master core that the client wants to use. This instance is passed in qapi_I2CM_Open().

Enumerator:

QAPI_I2C_INSTANCE_SE0_E GENI I²C controller 0.

7.1.4 Function Documentation

7.1.4.1 qapi_Status_t qapi_I2CM_Open (qapi_I2CM_Instance_t *Instance,* qapi_I2CM_Config_t ∗ *Config*)

Initializes the respective I²C master instance.

The API allocates resources for use by the client handle and the I²C master instance, and enables power to the I²C hardware instance.

Parameters

in	*Instance*	I²C master instance that the client intends to initialize.
in	*Config*	I²C master configuration.

Returns

QAPI_OK – Module was initialized successfully.
QAPI_I2CM_ERROR_INVALID_PARAM – Invalid instance or handle parameter.
QAPI_I2CM_ERROR_DEVICE_STATE – I2C instance is not opened.
QAPI_I2CM_ERROR_IC_COMP – I2C component type is incorrect.
QAPI_I2CM_ERROR_BOOTSTRAP_CFG_FAIL – Config I2C bootstrap failed.

QAPI_I2CM_ERROR_BUS_CLK_ENABLE_FAIL – Config I2C clock failed.

7.1.4.2 qapi_Status_t qapi_I2CM_Close (qapi_I2CM_Instance_t *Instance*)

De-initializes the I²C master instance.

The API releases any resources allocated by the qapi_I2CM_Open() API, and disables the power to the instance. In this API, the transfers that have not been completed are canceled and the transfer call back function is called.

Parameters

in	*Instance*	The I²C master instance to be closed.

Returns

QAPI_OK – I²C master driver was closed successfully.
QAPI_I2CM_ERROR_INVALID_PARAM – Invalid instance parameter.
QAPI_I2CM_ERROR_DEVICE_STATE – I2C instance is not opened.
QAPI_I2CM_ERROR_BOOTSTRAP_CFG_FAIL – Config I2C bootstrap failed.
QAPI_I2CM_ERROR_BUS_CLK_ENABLE_FAIL – Config I2C clock failed.

7.1.4.3 qapi_Status_t qapi_I2CM_Transfer (qapi_I2CM_Instance_t *Instance,* qapi_I2CM_Transfer_Config_t * *Config,* qapi_I2CM_Descriptor_t * *Desc,* uint32_t *NumDesc,* qapi_I2CM_Transfer_CB_t *CBFunction,* void * *CBParameter*)

Performs an I²C master transfer.

In case a transfer is already in progress by another client, this call queues the transfer. If the transfer returns a failure, the transfer has not been queued and no callback will occur. If the transfer returns QAPI_OK, the transfer has been queued and a further status of the transfer is only obtainable when the callback is called in non-blocking mode.

NOTE In general, if the client wants to queue multiple transfers, it could use an array of descriptors of type qapi_I2CM_Descriptor_t instead of calling the API multiple times. Although the I²C master driver supports queuing multiple transfers in non-blocking mode.

Parameters

in	*Instance*	The I²C master instance.
in	*Config*	Transfer configuration.
in	*Desc*	I²C master transfer descriptor. This can be an array of descriptors.
in	*NumDesc*	Number of descriptors in the descriptor array.
in	*CBFunction*	The callback function that is called at the completion of the transfer, occurs in non-blocking mode. The call must do minimal processing and must not call any API defined in this file. It should be NULL in blocking mode.
in	*CBParameter*	The context that the client passes here, is returned as is, in the callback function. It should be NULL in blocking mode.

Returns

QAPI_OK – I²C master transfer successful.
QAPI_I2CM_ERROR_INVALID_PARAM – One or more parameters are invalid.
QAPI_I2CM_ERROR_TRANSFER_BUSY – I²C master core is busy.
QAPI_I2CM_ERROR_DEVICE_STATE – I2C instance is not opened.
QAPI_I2CM_ERROR_TRANSFER_TIMEOUT – Transfer timed out.
QAPI_I2CM_ERROR_INPUT_FIFO_UNDER_RUN – Software reads from an empty Rx FIFO.
QAPI_I2CM_ERROR_INPUT_FIFO_OVER_RUN – Hardware writes to a full Rx FIFO.
QAPI_I2CM_ERROR_OUTPUT_FIFO_OVER_RUN – Software writes a new word to a full Tx FIFO.
QAPI_I2CM_ERROR_TRANSFER_FORCE_TERMINATED – Transfer abort or cancel request by software.

7.1.4.4 qapi_Status_t qapi_I2CM_Cancel_Transfer (qapi_I2CM_Instance_t *Instance*)

Cancels a transfer.

A transfer that has been initiated successfully by calling qapi_I2CM_Transfer() may be canceled. Based on the internal state of the transfer, this function either immediately cancels the transfer or ends the transfer at a later time.

Parameters

in	*Instance*	The I²C master instance.

Returns

QAPI_OK – Transfer is cancelled successfully.
QAPI_I2CM_ERROR_INVALID_PARAM – One or more parameters are invalid.
QAPI_I2CM_CANCEL_TRANSFER_INVALID – No transfer to cancel.
QAPI_I2CM_ERROR_CANCEL_TRANSFER_FAIL – Transfer cancel failed on the bus.

7.2 I2C error codes

Error codes returned by the I²C master controller API.

7.2.1 Define Documentation

7.2.1.1 #define QAPI_I2CM_ERROR __QAPI_ERROR(QAPI_MOD_I2C, 1)

Common error

7.2.1.2 #define QAPI_I2CM_ERROR_INVALID_PARAM __QAPI_ERROR(QAPI_MOD_I2C, 2)

Invalid input parameters.

7.2.1.3 #define QAPI_I2CM_ERROR_MEM_ALLOC __QAPI_ERROR(QAPI_MOD_I2C, 3)

Alloc memory failed.

7.2.1.4 #define QAPI_I2CM_ERROR_TRANSFER_BUSY __QAPI_ERROR(QAPI_MOD_I2C, 4)

Transaction busy.

7.2.1.5 #define QAPI_I2CM_ERROR_TRANSFER_TIMEOUT __QAPI_ERROR(QAPI_MOD_I2C, 5)

Transaction timeout in blocking mode.

7.2.1.6 #define QAPI_I2CM_ERROR_INPUT_FIFO_UNDER_RUN __QAPI_ERROR(QAPI_MOD_I2C, 6)

Software reads from an empty Rx FIFO.

7.2.1.7 #define QAPI_I2CM_ERROR_INPUT_FIFO_OVER_RUN __QAPI_ERROR(QAPI_MOD_I2C, 7)

Hardware writes to a full Rx FIFO.

7.2.1.8 #define QAPI_I2CM_ERROR_OUTPUT_FIFO_UNDER_RUN __QAPI_ERROR(QAPI_MOD_I2C, 8)

Software reads a new word from an empty Tx FIFO.

7.2.1.9 #define QAPI_I2CM_ERROR_OUTPUT_FIFO_OVER_RUN __QAPI_ERROR(QAPI_MOD_I2C, 9)

Software writes a new word to a full Tx FIFO.

7.2.1.10 #define QAPI_I2CM_ERROR_COMMAND_OVER_RUN __QAPI_ERROR(QAPI_MOD_I2C, 10)

A new command is initialized before the previous one is complete.

7.2.1.11 #define QAPI_I2CM_ERROR_TRANSFER_FORCE_TERMINATED __QAPI_ERROR(QAPI_MOD_I2C, 11)

Command abort, or cancel request by software.

7.2.1.12 #define QAPI_I2CM_ERROR_COMMAND_ILLEGAL __QAPI_ERROR(QAPI_MOD_I2C, 12)

Command with an illegal opcode.

7.2.1.13 #define QAPI_I2CM_ERROR_COMMAND_FAIL __QAPI_ERROR(QAPI_MOD_I2C, 13)

Command execution has been completed with a failure.

7.2.1.14 #define QAPI_I2CM_ERROR_BUS_CLK_ENABLE_FAIL __QAPI_ERROR(QAPI_MOD_I2C, 14)

Setting the clock failed.

7.2.1.15 #define QAPI_I2CM_ERROR_BUS_GPIO_ENABLE_FAIL __QAPI_ERROR(QAPI_MOD_I2C, 15)

Setting the GPIO failed.

7.2.1.16 #define QAPI_I2CM_ERROR_DMA_TX_BUS_ERROR __QAPI_ERROR(QAPI_MOD_I2C, 16)

Bus error during DMA Tx transaction.

7.2.1.17 #define QAPI_I2CM_ERROR_DMA_RX_BUS_ERROR __QAPI_ERROR(QAPI_MOD_I2C, 17)

Bus error during DMA Rx transaction.

7.2.1.18 #define QAPI_I2CM_DMA_TX_RESET_DONE __QAPI_ERROR(QAPI_MOD_I2C, 18)

DMA TX reset done.

7.2.1.19 #define QAPI_I2CM_DMA_RX_RESET_DONE __QAPI_ERROR(QAPI_MOD_I2C, 19)

DMA RX reset done.

7.2.1.20 #define QAPI_I2CM_CANCEL_TRANSFER_COMPLETED __QAPI_ERROR(QAPI_MOD_I2C, 20)

Transfer complete when canceled.

7.2.1.21 #define QAPI_I2CM_CANCEL_TRANSFER_INVALID __QAPI_ERROR(QAPI_MOD_I2C, 21)

No transfer to be canceled.

7.2.1.22 #define QAPI_I2CM_ERROR_CANCEL_TRANSFER_FAIL __QAPI_ERROR(QAPI_MOD_I2C, 22)

Transfer cancel failed

7.2.1.23 #define QAPI_I2CM_ERROR_BOOTSTRAP_CFG_FAIL __QAPI_ERROR(QAPI_MOD_I2C, 23)

Configure bootstrap for I2C module failed

7.2.1.24 #define QAPI_I2CM_ERROR_DEVICE_STATE __QAPI_ERROR(QAPI_MOD_I2C, 24)

I2C instance device state error

7.2.1.25 #define QAPI_I2CM_ERROR_INIT_XFR __QAPI_ERROR(QAPI_MOD_I2C, 25)

I2C initilize tranfer failed

7.2.1.26 #define QAPI_I2CM_ERROR_IC_COMP __QAPI_ERROR(QAPI_MOD_I2C, 26)

I2C component type error

7.2.1.27 #define QAPI_I2CM_ERROR_TX_ABORT_INTR __QAPI_ERRPR(QAPI_MOD_I2C, 27)

I2C received tx abort intrruption

7.3 I2C transfer flags

I²C transfer flags

7.3.1 Define Documentation

7.3.1.1 #define QAPI_I2C_FLAG_START 0x00000001

I²C transfer flags Specifies that the transfer begins with a START bit - S.

7.3.1.2 #define QAPI_I2C_FLAG_STOP 0x00000002

Specifies that the transfer ends with a STOP bit - P.

7.3.1.3 #define QAPI_I2C_FLAG_WRITE 0x00000004

Must be set to indicate a WRITE transfer.

7.3.1.4 #define QAPI_I2C_FLAG_READ 0x00000008

Must be set to indicate a READ transfer.

8 Low Power module

8.1 Low power APIs

8.1.1 Typedef Documentation

8.1.1.1 typedef bool(∗ qapi_bmps_rx_filter_cb)(uint16_t type, bool bm_cast, void ∗pbuf, uint16_t len)

BMPS Rx filter callback typedef.

8.1.2 Function Documentation

8.1.2.1 qapi_Status_t qapi_pm_enable (uint8_t *enable*)

Enables or disables system power management.

Parameters

in	*enable*	Enable system power management; **Supported values:** • 1 – Enable • 0 – Disable

Returns

- QAPI_OK – Enabled/disabled system power management successfully.

8.1.2.2 qapi_Status_t qapi_deepsleep_enter (uint8_t *wkup_src,* uint64_t *sleep_time*)

Puts the system into deep sleep directly.

Parameters

in	*wkup_src*	Wakeup source; **Supported values:** • 1 – AON timer • 2 – External wakup source
in	*sleep_time*	Sleep time in us, only valid when wkup_src is set 1

Returns

- QAPI_OK – System successfully entered deepsleep.

8.1.2.3 qapi_Status_t qapi_imps_cfg (uint8_t *enable,* uint32_t *sleep_time,* uint32_t *recnx_wait,* uint32_t *wmi_wait,* uint32_t *cnx_wait,* qapi_sleep_mode *policy*)

Configures and enables IMPS using deepsleep with AON timer as the wakeup source.

Parameters

in	*enable*	1: Enable; 0: disable. Other parameters are valid only when enable is 1.
in	*sleep_time*	Sleep time in ms, during deepsleep state.
in	*recnx_wait*	Re-connection timeout in ms. When WLAN disconnect/connect_fail happens, this timer will start. If successful connection occurs, cancel the timer. If timeout, system determines whether to enter into deepsleep.
in	*wmi_wait*	Wmi_wait time in ms. Upon recnx_wait timeout, check if there's any WMI cmd received during the wmi_wait duration. If no, go to deepsleep. If yes, start a timer with wmi_wait duration.
in	*cnx_wait*	Time in ms. Use for ENABLE_IMPS_TIMER_ON_BOOTUP feature. Indicates starting this timer during bootup. If there's no WLAN connection during this period, then system enters into deepsleep.

Returns

- QAPI_OK – IMPS cfg and enable successfully.

8.1.2.4 qapi_Status_t qapi_imps_enter_sleep (uint8_t *enable,* uint32_t *wait_time,* uint32_t *sleep_time*)

Configures and enables IMPS (using deep sleep with AON timer as wakeup source).

Parameters

in	*enable*	1: Enable; 0: disable. Other parameters are valid only when enable is 1.
in	*sleep_time*	Sleep time in ms, during deepsleep state;
in	*recnx_wait*	Re-connection timeout in ms. When wlan disconnect/connect_fail happens, this timer will start; if connect success happens then cancel the timer; if timeout, system will determine whether to enter into deepsleep.
in	*wmi_wait*	Wmi_wait time in ms. Upon recnx_wait timeout, check if there's any WMI cmd received during the wmi_wait duration, if no then goto deepsleep, if yes then start a timer with wmi_wait duration.
in	*cnx_wait*	Time in ms. Use for ENABLE_IMPS_TIMER_ON_BOOTUP feature, means starting this timer during bootup, if there's no wlan connection during this period, then system enters into deepsleep.

Returns

- QAPI_OK – IMPS configured and enabled successfully.

8.1.2.5 qapi_Status_t qapi_imps_disable_sleep (void)

Disables IMPS.

Returns

- QAPI_OK – IMPS Disenable successfully.

8.1.2.6 qapi_Status_t qapi_bmps_cfg (uint8_t *enable,* uint32_t *idle_timeout*)

Configures and enablee or disables BMPS.

Parameters

in	*enable*	**Supported values:** • 1 – Enable • 0 – disable
in	*idle_timeout*	Idle timeout value in ms. When BMPS is enabled, the system starts a timer with idle_timeout as the timeout value. After the timer expires, the system checks Tx/Rx count during this period. If condition are met, the system enters BMPS, otherwise it restarts the idle timer.

Returns

- QAPI_OK – BMPS configured and enabled/disabled successfully.

8.1.2.7 qapi_Status_t qapi_bmps_rx_filter_enable (uint8_t *enable*)

Configures and enables or disables the BMPS Rx filter.

Parameters

in	*enable*	**Supported values:** • 1: Enable – 0: disable

Returns

- QAPI_OK – BMPS Rx filter enabled or disabled successfully.

8.1.2.8 qapi_Status_t qapi_bmps_bcmc_rx_filter_cb_register (qapi_bmps_rx_filter_-cb *bmps_cb,* qapi_bmps_rx_filter_cb *net_cb*)

Registers the Rx filter callback function for broadcast/multicast packets.

Parameters

in	*bmps_cb*	Callback function used in BMPS mode; used as a filter to ignore some broadcast/multicast packets that will not wake up the chip.
in	*net_cb*	Callback function used in network stack; can be NULL;

Returns

- QAPI_OK – BMPS Rx filter enabled or disabled successfully.

8.1.2.9 qapi_Status_t qapi_bmps_sleep_wakeup_cb (ps_evt_cb_t *cb,* uint8_t *flag*)

Register the callback function for bmps for pre-sleep/post-awake.

Parameters

in	*cb*	callback function used for calling when bmps pre-sleep/post-awake.
in	*flag*	flag to register/deregister

Returns

- QAPI_OK – BMPS register callback successfully.

8.1.2.10 qapi_Status_t qapi_bmps_get_exit_reason (uint8_t ∗ *reason*)

Obtain exit reason of BMPS.

Parameters

in	∗*reason*	Pointer of exit reason to obtain

Returns

- QAPI_OK – valid pointer.

9 WLAN module

9.1 WLAN

9.1.1 Data Structure Documentation

9.1.1.1 struct qapi_WLAN_Start_Scan_Params_t

Data structure used by the application to pass wireless scan options to the driver. If the device is connected to an AP, a successful scan returns the current AP along with newly scanned APs.

Data fields

Type	Parameter	Description
int32_t	force_Fg_Scan	Force a high priority scan.
uint32_t	home_Dwell_-Time_In_Ms	Maximum scan duration in the home channel (in ms). If set to 0, the default value of 50 ms is used.
uint32_t	force_Scan_-Interval_In_Ms	Time interval (in ms) between scanning channels from the list. If set to 0, the default value of 100 ms is used.
uint8_t	scan_Type	This parameter currently supports only 0 as an input value.
uint8_t	num_Channels	Number of channels to scan.
uint16_t	channel_List[1]	List of channels to scan.
uint8_t	ssid[__QAPI_-WLAN_MAX-_SSID_LEN]	SSID.
uint8_t	ssid_Length	SSID length.

9.1.1.2 struct qapi_WLAN_BSS_Scan_Info_t

Data structure that the application uses to interpret the scan results for all access point information received during the scan.

All the information in this structure is for one particular BSS found during the scan.

Data fields

Type	Parameter	Description
uint8_t	channel	Wireless channel.
uint8_t	ssid_Length	SSID length.
uint8_t	rssi	Received signal strength indicator.
uint8_t	security_-Enabled	1: Security enabled; 0: Security disabled.
uint16_t	beacon_Period	Beacon period.
uint8_t	preamble	Preamble.
uint8_t	bss_type	BSS type.

Type	Parameter	Description
uint8_t	bssid[__QAPI_WLAN_MAC_LEN]	BSSID.
uint8_t	ssid[__QAPI_WLAN_MAX_SSID_LEN]	SSID.
uint8_t	rsn_Cipher	RSN cipher.
uint8_t	rsn_Auth	RSN authentication.
uint8_t	wpa_Cipher	WPA cipher.
uint8_t	wpa_Auth	WPS authentication.
uint16_t	caps	Capability IE.
uint8_t	wep_Support	Supprt for WEP.
uint8_t	reserved[3]	Reserved.

9.1.1.3 struct qapi_WLAN_Evt_Hdr_t

Overall status.

Data fields

Type	Parameter	Description
qapi_Status_t	status	Status.

9.1.1.4 struct qapi_WLAN_Enable_Evt_t

Data fields

Type	Parameter	Description
qapi_WLAN_Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	mac_addr[__QAPI_WLAN_MAC_LEN]	Assigned MAC address.
uint8_t	num_networks	Number of vdev supported.
uint8_t	reserved	Reserved.
uint32_t	cap_info	Capability info bitmap.
uint32_t	cap_info2	Additional capability info bitmap.
uint32_t	reserved2	Reserved.

9.1.1.5 struct qapi_WLAN_Disable_Evt_t

WLAN disable event.

Data fields

Type	Parameter	Description
qapi_WLAN_Evt_Hdr_t	evt_hdr	Contains the common event header.
uint32_t	reserved	Reserved.

9.1.1.6 struct qapi_WLAN_If_Add_Comp_Evt_t

WLAN interface add complete event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint32_t	reserved	Reserved.

9.1.1.7 struct qapi_WLAN_Scan_Start_Evt_t

WLAN scan start event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	scan_id	Identifier for a specific scan
uint8_t	reserved[3]	Reserved.

9.1.1.8 struct qapi_WLAN_Scan_Comp_Evt_t

WLAN scan complete event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	num_bss_cur	Number of BSS in current structure.
uint8_t	scan_id	Scan ID.
uint8_t	total_bss	
uint8_t	reserved2	Reserved.
qapi_WLAN-_BSS_Scan_-Info_t	scan_bss_-info[0]	BSS info array, count is num_bss_cur

9.1.1.9 struct qapi_WLAN_Join_Comp_Evt_t

WLAN join complete event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	bssid[__QAPI-_WLAN_MA-C_LEN]	BSSID of the connected access point.

Type	Parameter	Description
uint8_t	bss_-Connection_-Status	Flag that indicates whether it is a BSS level connection/disconnection or an individual station-level connection/disconnection. • STA mode connect event – A value of 1 indicates that the device is connected to an AP • STA mode disconnect event – A value of 1 indicates that the device is disconnected from an AP • STA mode – It is not expected to receive this value as 0 when operating in STA mode • AP mode connect Event – A value of 1 indicates that the SoftAP session has been started • AP mode disconnect event – A value of 1 indicates that the SoftAP session has been stopped • AP mode connect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has connected to SoftAP • AP Mode disconnect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has disconnected from SoftAP
uint8_t	ssid_Length	SSID length.
uint8_t	ssid[__QAPI_-WLAN_MAX-_SSID_LEN]	SSID of connected access point.
uint16_t	assoc_id	Association ID
uint8_t	host_initiated	Specifies whether connection is host-initiated or not.
uint8_t	reason_code	Reason code.
uint16_t	channel_-frequency	Channel frequency.
uint8_t	passphrase[__-QAPI_WLAN-_PASSPHRAS-E_LEN+1]	Passphrase of joined AP.
uint16_t	reserved2	Reserved.

9.1.1.10 struct qapi_WLAN_Chan_Switch_Evt_t

WLAN channel switch event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	reason	Reason code.
uint16_t	freq	
uint8_t	reserved	Channel frequency. Reserved.
uint8_t	reserved2	Reserved.

9.1.1.11 struct qapi_WLAN_WPS_Fail_Evt_t

WLAN WPS fail event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	reason	Reason code.
uint8_t	reserved[3]	Reserved .

9.1.1.12 struct qapi_WLAN_Connect_Cb_Info_t

Data structure that presents connect event information from the driver to the application.

The application uses this data structure to interpret the event payload received with a QAPI_WLAN_CONNECT_CB_E event.

Data fields

Type	Parameter	Description
int32_t	value	TRUE: To indicate connect events FALSE: To indicate disconnect/deauthorization events
uint8_t	mac_Addr[__-QAPI_WLAN-_MAC_LEN]	MAC address related to the connect event. Based on the operating mode, the driver fills in a different MAC addresses as follows. • STA mode connect event – MAC address of the connected access point • STA mode disconnect event – MAC address contains all zeroes • AP mode connect event – MAC address of the device itself, or a peer station that was connected to the AP (based on the bss_Connection_Status parameter) • AP mode disconnect event – MAC address contains zeros or a peer station that was disconnected from the AP (based on the bss_Connection_Status parameter)

Type	Parameter	Description
uint32_t	bss_-Connection_-Status	Flag that indicates whether it is a BSS level connection/disconnection or an individual station-level connection/disconnection. • STA mode connect event – A value of 1 indicates that the device is connected to an AP • STA mode disconnect event – A value of 1 indicates that the device is disconnected from an AP • STA mode – It is not expected to receive this value as 0 when operating in STA mode • AP mode connect Event – A value of 1 indicates that the SoftAP session has been started • AP mode disconnect event – A value of 1 indicates that the SoftAP session has been stopped • AP mode connect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has connected to SoftAP • AP Mode disconnect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has disconnected from SoftAP
int32_t	disConnReason	Flag that indicates disconnect reason code. It has the same value with WMI_DISCONNECT_REASON (defined in wmi.h). The following examples are expected, typical values: • NO_NETWORK_AVAIL = 0x01, • LOST_LINK = 0x02, • DISCONNECT_CMD = 0x03, • BSS_DISCONNECTED = 0x04,

9.1.1.13 struct qapi_WLAN_11n_HT_Config_t

Data structure to set 11n HT configurations.

Data fields

Type	Parameter	Description
qapi_WLA-N_11n_HT_-Config_e	htconfig	Enumeration that provides 11n HT configurations.
uint8_t	sgi	20M short GI enable flag.
uint8_t	mpdu_density	MPDU density (aka Minimum MPDU Start Spacing).

9.1.1.14 struct qapi_WLAN_Power_Mode_Params_t

Data structure used to request the necessary operating power mode (Maximum Performance or Power Save (rec_power) mode).

Data fields

Type	Parameter	Description
qapi_WLAN_-Power_Mode_e	power_Mode	Power mode to be set
qapi_WLAN_-Power_Module-_e	power_Module	Module requesting power mode change

9.1.1.15 struct qapi_WLAN_Add_Pattern_t

Data structure to add a pattern structure for WOW and filtering.

Data fields

Type	Parameter	Description
uint32_t	pattern_Index	Pattern index.
uint32_t	pattern_Action-_Flag	Pattern action flag.
uint32_t	offset	Offset.
uint32_t	pattern_Size	Pattern size.
uint16_t	header_Type	Header type.
uint8_t	pattern_Priority	Pattern priority.
uint8_t	pattern_Mask[-__QAPI_WLA-N_PATTERN-_MASK]	Pattern mask.
uint8_t	pattern[__QAP-I_WLAN_PA-TTERN_MAX-_SIZE]	Pattern string.

9.1.1.16 struct qapi_WLAN_Promiscuous_Mode_Info_t

Data structure to configure a device in Promiscuous mode with the necessary filters, if required. Promiscuous mode is only supported in virtual device 0. The maximum number of filters supported is __QAPI_WLAN_PROMISC_MAX_FILTER_IDX. All the necessary filters should be set at once before passing this data structure to qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE as the command ID.

Dependencies

The device should be set to Maximum Performance mode before enabling Promiscuous mode.

Data fields

Type	Parameter	Description
uint8_t	src_Mac[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX][__QAPI_WLAN_MAC_LEN]	Array of source MAC addresses that are of interest to the application. The first array index here corresponds to the filter index and the second index of the array holds the actual source MAC address on which incoming 802.11 frames are to be filtered by the firmware.
uint8_t	dst_Mac[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX][__QAPI_WLAN_MAC_LEN]	Array of destination MAC addresses that are of interest to the application. The first array index here corresponds to the filter index and the second index of the array holds the actual destination MAC address on which incoming 802.11 frames are to be filtered by the firmware.
uint8_t	enable	Promiscuous mode control; 1: Enable, 0: Disable.
uint8_t	filter_flags[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]	Filter flags that represent the validity of filters being programmed. Promiscuous mode supports filters based on four factors: source MAC, destination MAC, frame type, and frame subtype. The application must set the corresponding bit in this field for each of the filter indexes to indicate which of the configured filters are valid: • Set Bit 0 if the source MAC address filter is valid • Set Bit 1 if the destination MAC address filter is valid • Set Bit 2 if the frame type filter is valid • Set Bit 3 if the frame subtype filter is valid
uint8_t	promisc_frametype[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]	Frame type based filter. The application can choose whether it is interested in an 802.11 Control/Data/Management packet type. The application can only choose one of these packet types for a given filter index.
uint8_t	promisc_subtype[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]	Frame subtype based filter. The application can choose to filter frames of a given subtype. This subtype filter is valid only if the frame type filter is valid in the corresponding filter index. In other words, if a subtype filter is set on a filter index, but the corresponding frame type filter is not set, the subtype filter is not considered valid.
uint8_t	promisc_num_filters	Number of programmed promiscuous filters. This should not exceed __QAPI_WLAN_PROMISC_MAX_FILTER_IDX.

9.1.1.17 struct qapi_WLAN_Raw_Send_Params_t

Data structure that the application is to pass when invoking qapi_WLAN_Raw_Send() to transmit a raw frame.

Data fields

Type	Parameter	Description
uint8_t	rate_Index	0: 1 Mbps, 1: 2 Mbps, 2: 5.5 Mbps, etc. Note that this value will not take effect if connected
uint8_t	num_Tries	Packet transmission count: 1 to 14.
uint32_t	payload_Size	Payload size: 0 to 1400.
uint32_t	channel	Channel; 0 to 11. 0: Send on the current channel. Note that this value will not take effect if connected
qapi_WLAN-_Raw_Mode_-Header_Type_e	header_Type	0: Beacon frame, 1: Probe Request frame, 2: QoS data frame, 3: Four addresses data frame, 0xff: Self Defined frame If use Self Defined Header, the frame will be considered as an MGMT frame.>
uint16_t	seq	Sequence number to be filled in the 802.11 header.
uint8_t	addr1[__QAPI-_WLAN_MA-C_LEN]	Address 1.
uint8_t	addr2[__QAPI-_WLAN_MA-C_LEN]	Address 2.
uint8_t	addr3[__QAPI-_WLAN_MA-C_LEN]	Address 3.
uint8_t	addr4[__QAPI-_WLAN_MA-C_LEN]	Address 4. Note that address will not take effect when using self-defined frame
uint32_t	data_Length	Size of the data to be transmitted.
uint8_t *	data	Data.

9.1.1.18 struct qapi_WLAN_WPS_Credentials_t

Data structure to pass WPS credentials from the driver to the application.

Data fields

Type	Parameter	Description
uint16_t	ap_Channel	AP's channel.
uint8_t	ssid[__QAPI_-WLAN_MAX-_SSID_LEN]	SSID.
uint8_t	ssid_Length	SSID length.
qapi_WLAN_-Auth_Mode_e	auth_Mode	Authentication mode.
qapi_WLAN_-Crypt_Type_e	encryption_-Type	Encryption type.
uint8_t	key_Index	Index of the key.
uint8_t	key[__QAPI_-WLAN_WPS_-MAX_KEY_L-EN+1]	Key.
uint8_t	key_Length	Key length.

Type	Parameter	Description
uint8_t	mac_Addr[__QAPI_WLAN_MAC_LEN]	MAC address.

9.1.1.19 struct qapi_WLAN_Aggregation_Params_t

Data structure for A-MPDU enablement. A-MPDU aggregation is enabled on a per-TID basis, where each TID (0-7) represents a different traffic priority.

The mapping to WMM access categories is as follows:

- WMM best effort = TID 0-3
- WMM background = TID 1-2
- WMM video = TID 4-5
- WMM voice = TID 6-7

Once enabled, A-MPDU aggregation may be negotiated with an access point/Peer device and then both devices may optionally use A-MPDU aggregation for transmission. Due to other bottle necks in the data path, a system may not get improved performance by enabling A-MPDU aggregation.

Data fields

Type	Parameter	Description
uint16_t	tx_TID_Mask	Bitmask to enable Tx A-MPDU aggregation.
uint16_t	rx_TID_Mask	Bitmask to enable Rx A-MPDU aggregation.

9.1.1.20 struct qapi_WPS_Scan_List_Entry_t

Data structure used to pass WPS SSID information from the application to the driver.

Data fields

Type	Parameter	Description
uint8_t	ssid[__QAPI_WLAN_MAX_SSID_LEN]	SSID.
uint8_t	macaddress[__QAPI_WLAN_MAC_LEN]	MAC address.
uint16_t	channel	Wireless channel.
uint8_t	ssid_Len	SSID length.

9.1.1.21 struct qapi_WLAN_WPS_Start_t

Data structure to pass WPS command parameters from the application to the driver.

Data fields

Type	Parameter	Description
qapi_WPS_-Scan_List_-Entry_t	ssid_info	SSID information.
uint8_t	wps_Mode	WPS pushbutton or WPS PIN.
uint8_t	timeout_-Seconds	WPS timeout in seconds.
uint8_t	connect_Flag	Connect action (TRUE/FALSE) after intial WPS success.
uint8_t	pin[__QAPI_-WLAN_WPS_-PIN_LEN]	PIN.
uint8_t	pin_Length	PIN length.

9.1.1.22 struct qapi_WLAN_Cipher_t

Data structure for cipher.

Data fields

Type	Parameter	Description
uint32_t	ucipher	Unicast cipher.
uint32_t	mcipher	Multicast cipher.

9.1.1.23 struct qapi_WLAN_Netparams_t

Data structure for wireless network parameters.

Data fields

Type	Parameter	Description
uint16_t	ap_Channel	Wireless channel for the AP.
int8_t	ssid[__QAPI_-WLAN_MAX-_SSID_LENG-TH]	[OUT] Network SSID.
int16_t	ssid_Len	[OUT] Number of valid chars in ssid[].
qapi_WLAN_-Cipher_t	cipher	[OUT] Network cipher type values not defined.
uint8_t	key_Index	[OUT] For WEP only; key index for Tx.
union qapi-_WLAN_-Netparams_t	u	[OUT] Security key or passphrase.
uint8_t	sec_Type	[OUT] Security type.
uint8_t	error	[OUT] Error code.
uint8_t	dont_Block	[IN] 1 – Returns immediately if the operation is not complete 0 – Blocks until the operation completes

9.1.1.24 union qapi_WLAN_Netparams_t.u

Data fields

Type	Parameter	Description
uint8_t	wepkey[__QAPI_WLAN_PASSPHRASE_LEN+1]	WEP key.
uint8_t	passphrase[__QAPI_WLAN_PASSPHRASE_LEN+1]	Passphrase.

9.1.1.25 struct qapi_WLAN_App_Ie_Params_t

Data structure to pass application information element data from the application to the driver.

Data fields

Type	Parameter	Description
uint8_t	mgmt_Frame_Type	Frame in which IE is to be added.
uint8_t	ie_Len	IE length.
uint8_t *	ie_Info	Application specified IE.

9.1.1.26 struct qapi_WLAN_Set_Txpower_Params_t

Data structure for Tx Power.

Data fields

Type	Parameter	Description
uint8_t	txpower	
qapi_WLAN_TX_Power_Policy_e	policy	

9.1.1.27 struct qapi_WLAN_Reg_t

Data structure used by the application to interpret the regulatory information the device supports.

All information in this structure is for one regulatory event supported by the device.

Data fields

Type	Parameter	Description
uint16_t	start_freq	Start frequency.
uint16_t	end_freq	End frequency.
uint8_t	reg_power	Regulatory power.
uint8_t	ant_gain	Antenna gain.
uint16_t	flag_info	Flag information.

Type	Parameter	Description
uint16_t	max_bw	Maximum bandwidth

9.1.1.28 struct qapi_WLAN_Reg_Evt_t

Data structure used by the application to interpret regulatory information events.

Data fields

Type	Parameter	Description
uint8_t	alpha[3]	Alpha.
uint8_t	num_2g_reg_-rules	2G regulatory rules.
uint8_t	num_5g_reg_-rules	5G regulatory rules.
qapi_WLAN_-Reg_t	reg_rules[QAP-I_MAX_REG-_RULES]	Data structure used by the application to interpret the regulatory information the device supports.

9.1.1.29 struct qapi_WLAN_Get_Power_Evt_t

Data fields

Type	Parameter	Description
uint8_t	reg_power	
uint8_t	ctl_power	Regulatory powe. r
uint16_t	target_power	Conformance test limit power.
uint16_t	real_power	The targeted maximum TX power.

9.1.1.30 struct qapi_WLAN_Set_Rate_Params_t

Set Rate.

Data fields

Type	Parameter	Description
uint8_t	ra_ON	Flag indicating whether automatic rate adaptation is enabled. Use NT_RA_OFF to disable or NT_RA_ON to enable.
uint8_t	rate_staid	Station ID for which the rate configuration applies.
uint8_t	rate_p_rate	Primary transmission rate. If transmission fails NT_DEVCFG_RETRY_THRESHOLD0 times (configurable; see rate_adaptation_debug.xml), the secondary rate will be attempted.
uint8_t	rate_s_rate	Secondary transmission rate. If transmission fails an additional (NT_DEVCFG_RETRY_THRESHOLD1 - NT_DEVCFG_RETR-Y_THRESHOLD0) times, the tertiary rate will be attempted.
uint8_t	rate_t_rate	

9.1.1.31 struct qapi_WLAN_Listen_Interval_Params_t

Set STA Listen interval.

Data fields

Type	Parameter	Description
uint32_t	time	Listen interval in Time Units (TU), where 1 TU = 1024 microseconds.
uint32_t	round_type	Determines how the beacon interval is rounded when calculating (time / beacon interval).

9.1.1.32 struct qapi_WLAN_Edca_Params_t

Set STA edca param, including aifsn/cw_min/cw_max/txoplimit.

Data fields

Type	Parameter	Description
uint8_t	qid	Queue ID∗./ uint8_t aifsn; /∗∗< Arbitration inter frame spacing number.
uint16_t	cw_min	Minimum value of contention window.
uint16_t	cw_max	Maximum value of contention window.
uint16_t	txop_limit	Transmission opportunity limit.

9.1.1.33 struct qapi_WLAN_BA_Window_Params_t

Set STA BA window parameters.

Data fields

Type	Parameter	Description
uint16_t	ack_timeout	BA window ack timeout in us.
uint16_t	delay	Propagation time in numbers of SM clock cycles.

9.1.1.34 struct qapi_WLAN_BA_Window_Size_t

Set STA BA window size.

Data fields

Type	Parameter	Description
uint16_t	tx_size	TX BA window size.
uint16_t	rx_size	RX BA window size.

9.1.2 Enumeration Type Documentation

9.1.2.1 enum qapi_WLAN_Scan_Status_e

This data type enumerates the scan status to indicate the scan is succeeded, required to rescan or failed.

9.1.2.2 enum qapi_WLAN_Callback_ID_e

WLAN driver invokes an application-registered callback function to indicate various asynchronous events to the application. This data structure enumerates the list of various event IDs for which the WLAN driver invokes the application-registered callback function.

Enumerator:

QAPI_WLAN_CONNECT_CB_E ID to indicate connect/disconnect events.

QAPI_WLAN_SCAN_COMPLETE_CB_E ID to indicate a wireless scan complete event, received for nonblocking/nonbuffering scans.

QAPI_WLAN_FWD_PROBE_REQUEST_INFO_CB_E ID to indicate a probe request forwarded from the WLAN firmware when probe request forwarding is enabled by the application.

QAPI_WLAN_RESUME_CB_INFO_CB_E ID to indicate a WLAN driver/firmware resume completed event and that the application can start sending data over the WLAN driver after receiving this event (for future use).

QAPI_WLAN_PROMISCUOUS_MODE_CB_INFO_CB_E ID to indicate that packets were captured in Promiscuous mode. For every packet received in Promiscuous mode, the driver indicates this event to the application. Since it is quite possible to receive packets too frequently in Promiscuous mode, the application is expected to do a minimal operation in the callback implementation.

QAPI_WLAN_DISCONNECT_CB_E Currently not used. A disconnect event is indicated in the parameter of QAPI_WLAN_CONNECT_CB_E.

QAPI_WLAN_DEAUTH_CB_E Currently not used. A disconnect event is indicated in the parameter of QAPI_WLAN_CONNECT_CB_E.

QAPI_WLAN_WPS_CB_E ID to indication completion of a WPS handshake to the application. The application should use qapi_WLAN_WPS_Await_Completion() to get event information.

QAPI_WLAN_P2P_CB_E ID to indicate all P2P events. Since most of the P2P event handling is done by the application, it is necessary for the application to copy necessary information from events(in an event callback) and handle this in the application's own thread context.

QAPI_WLAN_BSS_INFO_CB_E ID to indicate that AP profile information was received when performing a nonbuffering scan. Each event of this type indicates only the AP's profile information.

QAPI_WLAN_TCP_KEEPALIVE_OFFLOAD_CB_E ID to indicate an event for a TCP Keepalive Offload request, received either on termination of TCP KA offload or when a TCP timeout occurs for one or more of the TCP KA sessions.

QAPI_WLAN_PREFERRED_NETWORK_OFFLOAD_CB_E ID to indicate that a PNO profile event was received when a profile match is found or when PNO is disabled by the application.

QAPI_WLAN_WNM_CB_E ID to indicate events received when WNM commands, such as setting BSS maximum idle period, enter/exit WNM sleep complete execution.

QAPI_WLAN_CHANNEL_SWITCH_CB_E ID to indicate a wireless channel change event received when STA changes the operating channel due to a channel switch announcement IE from the connected access point.

QAPI_WLAN_READY_CB_E ID to indicate the event that announces the Wi-Fi module (firmware) is enabled and ready.

QAPI_WLAN_SUSPEND_CB_E ID to indicate a WLAN firmware suspend event.

QAPI_WLAN_DRIVER_DISABLE_CB_E ID to indicate a driver shut down complete event.

QAPI_WLAN_ERROR_HANDLER_CB_E ID to indicate a fatal error in the WLAN driver.

QAPI_WLAN_RESUME_HANDLER_CB_E ID to indicate to resume completion of the WLAN firmware.

QAPI_WLAN_RX_EAPOL_KEY_CB_E ID to indicate to receive eapol key frame.

QAPI_WLAN_RX_MGMT_CB_E ID to indicate to receive MGMT frame.

QAPI_WLAN_PMF_EVENT_CB_E ID to indicate PMF event.

QAPI_WLAN_SAE_COMMIT_CB_E ID to indicate to prepare SAE auth commit frame.

QAPI_WLAN_COUNTRY_CODE_CB_E ID to indicate country code is set.

QAPI_WLAN_ENABLE_CB_E ID to indicate WLAN is enabled.

QAPI_WLAN_DISABLE_CB_E ID to indicate WLAN is disabled.

QAPI_WLAN_IF_ADD_COMP_CB_E ID to indicate WLAN interface is added.

QAPI_WLAN_SCAN_START_CB_E ID to indicate WLAN scan is started.
QAPI_WLAN_WPS_FAIL_CB_E ID to indicate WLAN WPS failed.

9.1.2.3 enum qapi_WLAN_MGMT_FRAME_e

Identifies the management frame type.

Enumerator:

QAPI_WLAN_MGMT_NONE_E None.
QAPI_WLAN_MGMT_ASSOC_RESP_E Association response.
QAPI_WLAN_MGMT_PROBE_RESP_E Probe response.

9.1.3 Function Documentation

9.1.3.1 qapi_Status_t qapi_WLAN_Enable_Mgmt_Filter (uint8_t *device_ID,* uint32_t *mgmt_filter*)

API to be used to enable wlan management frame filter.

Parameters

in	*device_id*	Device ID.
in	*mgmt_filter*	WLAN management frames filter. Now only support association response and probe response frames.

Returns

qapi_Status_t QAPI_OK on success, other error code on failure.

9.1.3.2 qapi_Status_t qapi_WLAN_Disable_Mgmt_Filter (uint8_t *device_ID*)

API to be used to disable wlan management frame filter.

Parameters

in	*device_id*	Device ID.

Returns

qapi_Status_t QAPI_OK on success, other error code on failure.

9.1.3.3 qapi_Status_t qapi_WLAN_Recv_Mgmt_Frames (uint8_t * *buffer,* uint32_t *buffer_len,* uint32_t * *frame_len,* uint32_t *timeout*)

API to be used to recieve wlan management frames which are in filter.

Parameters

out	*buffer*	Pointer to output buffer for management frames .
in	*buffer_len*	Buffer lenght in bytes.
out	*frame_len*	lenght of management frames.

in	*timeout*	timeout in millisecond when receive management frames .

Returns

qapi_Status_t QAPI_OK on success, other error code on failure.

9.1.4 Variable Documentation

9.1.4.1 int32_t qapi_WLAN_Start_Scan_Params_t::force_Fg_Scan

Force a high priority scan.

9.1.4.2 uint32_t qapi_WLAN_Start_Scan_Params_t::home_Dwell_Time_In_Ms

Maximum scan duration in the home channel (in ms). If set to 0, the default value of 50 ms is used.

9.1.4.3 uint32_t qapi_WLAN_Start_Scan_Params_t::force_Scan_Interval_In_Ms

Time interval (in ms) between scanning channels from the list. If set to 0, the default value of 100 ms is used.

9.1.4.4 uint8_t qapi_WLAN_Start_Scan_Params_t::scan_Type

This parameter currently supports only 0 as an input value.

9.1.4.5 uint8_t qapi_WLAN_Start_Scan_Params_t::num_Channels

Number of channels to scan.

9.1.4.6 uint16_t qapi_WLAN_Start_Scan_Params_t::channel_List[1]

List of channels to scan.

9.1.4.7 uint8_t qapi_WLAN_Start_Scan_Params_t::ssid[__QAPI_WLAN_MAX_SSID_LEN]

SSID.

9.1.4.8 uint8_t qapi_WLAN_Start_Scan_Params_t::ssid_Length

SSID length.

9.1.4.9 uint8_t qapi_WLAN_BSS_Scan_Info_t::channel

Wireless channel.

9.1.4.10 uint8_t qapi_WLAN_BSS_Scan_Info_t::ssid_Length

SSID length.

9.1.4.11 uint8_t qapi_WLAN_BSS_Scan_Info_t::rssi

Received signal strength indicator.

9.1.4.12 uint8_t qapi_WLAN_BSS_Scan_Info_t::security_Enabled

1: Security enabled; 0: Security disabled.

9.1.4.13 uint16_t qapi_WLAN_BSS_Scan_Info_t::beacon_Period

Beacon period.

9.1.4.14 uint8_t qapi_WLAN_BSS_Scan_Info_t::preamble

Preamble.

9.1.4.15 uint8_t qapi_WLAN_BSS_Scan_Info_t::bss_type

BSS type.

9.1.4.16 uint8_t qapi_WLAN_BSS_Scan_Info_t::bssid[__QAPI_WLAN_MAC_LEN]

BSSID.

9.1.4.17 uint8_t qapi_WLAN_BSS_Scan_Info_t::ssid[__QAPI_WLAN_MAX_SSID_LEN]

SSID.

9.1.4.18 uint8_t qapi_WLAN_BSS_Scan_Info_t::rsn_Cipher

RSN cipher.

9.1.4.19 uint8_t qapi_WLAN_BSS_Scan_Info_t::rsn_Auth

RSN authentication.

9.1.4.20 uint8_t qapi_WLAN_BSS_Scan_Info_t::wpa_Cipher

WPA cipher.

9.1.4.21 uint8_t qapi_WLAN_BSS_Scan_Info_t::wpa_Auth

WPS authentication.

9.1.4.22 uint16_t qapi_WLAN_BSS_Scan_Info_t::caps

Capability IE.

9.1.4.23 uint8_t qapi_WLAN_BSS_Scan_Info_t::wep_Support

Supprt for WEP.

9.1.4.24 uint8_t qapi_WLAN_BSS_Scan_Info_t::reserved[3]

Reserved.

9.1.4.25 qapi_Status_t qapi_WLAN_Evt_Hdr_t::status

Status.

9.1.4.26 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Enable_Evt_t::evt_hdr

Contains the common event header.

9.1.4.27 uint8_t qapi_WLAN_Enable_Evt_t::mac_addr[__QAPI_WLAN_MAC_LEN]

Assigned MAC address.

9.1.4.28 uint8_t qapi_WLAN_Enable_Evt_t::num_networks

Number of vdev supported.

9.1.4.29 uint8_t qapi_WLAN_Enable_Evt_t::reserved

Reserved.

9.1.4.30 uint32_t qapi_WLAN_Enable_Evt_t::cap_info

Capability info bitmap.

9.1.4.31 uint32_t qapi_WLAN_Enable_Evt_t::cap_info2

Additional capability info bitmap.

9.1.4.32 uint32_t qapi_WLAN_Enable_Evt_t::reserved2

Reserved.

9.1.4.33 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Disable_Evt_t::evt_hdr

Contains the common event header.

9.1.4.34 uint32_t qapi_WLAN_Disable_Evt_t::reserved

Reserved.

9.1.4.35 qapi_WLAN_Evt_Hdr_t qapi_WLAN_If_Add_Comp_Evt_t::evt_hdr

Contains the common event header.

9.1.4.36 uint32_t qapi_WLAN_If_Add_Comp_Evt_t::reserved

Reserved.

9.1.4.37 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Scan_Start_Evt_t::evt_hdr

Contains the common event header.

9.1.4.38 uint8_t qapi_WLAN_Scan_Start_Evt_t::scan_id

Identifier for a specific scan

9.1.4.39 uint8_t qapi_WLAN_Scan_Start_Evt_t::reserved[3]

Reserved.

9.1.4.40 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Scan_Comp_Evt_t::evt_hdr

Contains the common event header.

9.1.4.41 uint8_t qapi_WLAN_Scan_Comp_Evt_t::num_bss_cur

Number of BSS in current structure.

9.1.4.42 uint8_t qapi_WLAN_Scan_Comp_Evt_t::scan_id

Scan ID.

9.1.4.43 uint8_t qapi_WLAN_Scan_Comp_Evt_t::total_bss

9.1.4.44 uint8_t qapi_WLAN_Scan_Comp_Evt_t::reserved2

Reserved.

9.1.4.45 qapi_WLAN_BSS_Scan_Info_t qapi_WLAN_Scan_Comp_Evt_t::scan_bss_info[0]

BSS info array, count is num_bss_cur

9.1.4.46 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Join_Comp_Evt_t::evt_hdr

Contains the common event header.

9.1.4.47 uint8_t qapi_WLAN_Join_Comp_Evt_t::bssid[__QAPI_WLAN_MAC_LEN]

BSSID of the connected access point.

9.1.4.48 uint8_t qapi_WLAN_Join_Comp_Evt_t::bss_Connection_Status

Flag that indicates whether it is a BSS level connection/disconnection or an individual station-level connection/disconnection.

- STA mode connect event – A value of 1 indicates that the device is connected to an AP
- STA mode disconnect event – A value of 1 indicates that the device is disconnected from an AP
- STA mode – It is not expected to receive this value as 0 when operating in STA mode
- AP mode connect Event – A value of 1 indicates that the SoftAP session has been started
- AP mode disconnect event – A value of 1 indicates that the SoftAP session has been stopped
- AP mode connect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has connected to SoftAP
- AP Mode disconnect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has disconnected from SoftAP

9.1.4.49 uint8_t qapi_WLAN_Join_Comp_Evt_t::ssid_Length

SSID length.

9.1.4.50 uint8_t qapi_WLAN_Join_Comp_Evt_t::ssid[__QAPI_WLAN_MAX_SSID_LEN]

SSID of connected access point.

9.1.4.51 uint16_t qapi_WLAN_Join_Comp_Evt_t::assoc_id

Association ID

9.1.4.52 uint8_t qapi_WLAN_Join_Comp_Evt_t::host_initiated

Specifies whether connection is host-initiated or not.

9.1.4.53 uint8_t qapi_WLAN_Join_Comp_Evt_t::reason_code

Reason code.

9.1.4.54 uint16_t qapi_WLAN_Join_Comp_Evt_t::channel_frequency

Channel frequency.

9.1.4.55 uint8_t qapi_WLAN_Join_Comp_Evt_t::passphrase[__QAPI_WLAN_PASSPHRASE_LEN+1]

Passphrase of joined AP.

9.1.4.56 uint16_t qapi_WLAN_Join_Comp_Evt_t::reserved2

Reserved.

9.1.4.57 qapi_WLAN_Evt_Hdr_t qapi_WLAN_Chan_Switch_Evt_t::evt_hdr

Contains the common event header.

9.1.4.58 uint8_t qapi_WLAN_Chan_Switch_Evt_t::reason

Reason code.

9.1.4.59 uint16_t qapi_WLAN_Chan_Switch_Evt_t::freq

9.1.4.60 uint8_t qapi_WLAN_Chan_Switch_Evt_t::reserved

Channel frequency. Reserved.

9.1.4.61 uint8_t qapi_WLAN_Chan_Switch_Evt_t::reserved2

Reserved.

9.1.4.62 qapi_WLAN_Evt_Hdr_t qapi_WLAN_WPS_Fail_Evt_t::evt_hdr

Contains the common event header.

9.1.4.63 uint8_t qapi_WLAN_WPS_Fail_Evt_t::reason

Reason code.

9.1.4.64 uint8_t qapi_WLAN_WPS_Fail_Evt_t::reserved[3]

Reserved .

9.1.4.65 int32_t qapi_WLAN_Connect_Cb_Info_t::value

TRUE: To indicate connect events

FALSE: To indicate disconnect/deauthorization events

9.1.4.66 uint8_t qapi_WLAN_Connect_Cb_Info_t::mac_Addr[__QAPI_WLAN_MAC_LEN]

MAC address related to the connect event. Based on the operating mode, the driver fills in a different MAC addresses as follows.

- STA mode connect event – MAC address of the connected access point
- STA mode disconnect event – MAC address contains all zeroes
- AP mode connect event – MAC address of the device itself, or a peer station that was connected to the AP (based on the bss_Connection_Status parameter)
- AP mode disconnect event – MAC address contains zeros or a peer station that was disconnected from the AP (based on the bss_Connection_Status parameter)

9.1.4.67 uint32_t qapi_WLAN_Connect_Cb_Info_t::bss_Connection_Status

Flag that indicates whether it is a BSS level connection/disconnection or an individual station-level connection/disconnection.

- STA mode connect event – A value of 1 indicates that the device is connected to an AP
- STA mode disconnect event – A value of 1 indicates that the device is disconnected from an AP
- STA mode – It is not expected to receive this value as 0 when operating in STA mode
- AP mode connect Event – A value of 1 indicates that the SoftAP session has been started
- AP mode disconnect event – A value of 1 indicates that the SoftAP session has been stopped
- AP mode connect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has connected to SoftAP
- AP Mode disconnect event – A value of 0 indicates that a peer station with a MAC address in mac_Addr has disconnected from SoftAP

9.1.4.68 int32_t qapi_WLAN_Connect_Cb_Info_t::disConnReason

Flag that indicates disconnect reason code. It has the same value with WMI_DISCONNECT_REASON (defined in wmi.h). The following examples are expected, typical values:

- NO_NETWORK_AVAIL = 0x01,
- LOST_LINK = 0x02,
- DISCONNECT_CMD = 0x03,
- BSS_DISCONNECTED = 0x04,

9.1.4.69 qapi_WLAN_11n_HT_Config_e qapi_WLAN_11n_HT_Config_t::htconfig

Enumeration that provides 11n HT configurations.

9.1.4.70 uint8_t qapi_WLAN_11n_HT_Config_t::sgi

20M short GI enable flag.

9.1.4.71 uint8_t qapi_WLAN_11n_HT_Config_t::mpdu_density

MPDU density (aka Minimum MPDU Start Spacing).

9.1.4.72 qapi_WLAN_Power_Mode_e qapi_WLAN_Power_Mode_Params_t::power_Mode

Power mode to be set

9.1.4.73 qapi_WLAN_Power_Module_e qapi_WLAN_Power_Mode_Params_t::power_Module

Module requesting power mode change

9.1.4.74 uint32_t qapi_WLAN_Add_Pattern_t::pattern_Index

Pattern index.

9.1.4.75 uint32_t qapi_WLAN_Add_Pattern_t::pattern_Action_Flag

Pattern action flag.

9.1.4.76 uint32_t qapi_WLAN_Add_Pattern_t::offset

Offset.

9.1.4.77 uint32_t qapi_WLAN_Add_Pattern_t::pattern_Size

Pattern size.

9.1.4.78 uint16_t qapi_WLAN_Add_Pattern_t::header_Type

Header type.

9.1.4.79 uint8_t qapi_WLAN_Add_Pattern_t::pattern_Priority

Pattern priority.

9.1.4.80 uint8_t qapi_WLAN_Add_Pattern_t::pattern_Mask[__QAPI_WLAN_PATTERN_MASK]

Pattern mask.

9.1.4.81 uint8_t qapi_WLAN_Add_Pattern_t::pattern[__QAPI_WLAN_PATTERN_MAX_SIZE]

Pattern string.

9.1.4.82 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::src_Mac[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX][__QAPI_WLAN_MAC_LEN]

Array of source MAC addresses that are of interest to the application. The first array index here corresponds to the filter index and the second index of the array holds the actual source MAC address on which incoming 802.11 frames are to be filtered by the firmware.

9.1.4.83 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::dst_Mac[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX][__QAPI_WLAN_MAC_LEN]

Array of destination MAC addresses that are of interest to the application. The first array index here corresponds to the filter index and the second index of the array holds the actual destination MAC address on which incoming 802.11 frames are to be filtered by the firmware.

9.1.4.84 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::enable

Promiscuous mode control; 1: Enable, 0: Disable.

9.1.4.85 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::filter_flags[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]

Filter flags that represent the validity of filters being programmed. Promiscuous mode supports filters based on four factors: source MAC, destination MAC, frame type, and frame subtype. The application must set the corresponding bit in this field for each of the filter indexes to indicate which of the configured filters are valid:

- Set Bit 0 if the source MAC address filter is valid
- Set Bit 1 if the destination MAC address filter is valid
- Set Bit 2 if the frame type filter is valid
- Set Bit 3 if the frame subtype filter is valid

9.1.4.86 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::promisc_frametype[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]

Frame type based filter. The application can choose whether it is interested in an 802.11 Control/Data/-Management packet type. The application can only choose one of these packet types for a given filter index.

9.1.4.87 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::promisc_subtype[__QAPI_WLAN_PROMISC_MAX_FILTER_IDX]

Frame subtype based filter. The application can choose to filter frames of a given subtype. This subtype filter is valid only if the frame type filter is valid in the corresponding filter index. In other words, if a subtype filter is set on a filter index, but the corresponding frame type filter is not set, the subtype filter is not considered valid.

9.1.4.88 uint8_t qapi_WLAN_Promiscuous_Mode_Info_t::promisc_num_filters

Number of programmed promiscuous filters. This should not exceed __QAPI_WLAN_PROMISC_MAX_FILTER_IDX.

9.1.4.89 uint8_t qapi_WLAN_Raw_Send_Params_t::rate_Index

0: 1 Mbps, 1: 2 Mbps, 2: 5.5 Mbps, etc. Note that this value will not take effect if connected

9.1.4.90 uint8_t qapi_WLAN_Raw_Send_Params_t::num_Tries

Packet transmission count: 1 to 14.

9.1.4.91 uint32_t qapi_WLAN_Raw_Send_Params_t::payload_Size

Payload size: 0 to 1400.

9.1.4.92 uint32_t qapi_WLAN_Raw_Send_Params_t::channel

Channel; 0 to 11. 0: Send on the current channel. Note that this value will not take effect if connected

9.1.4.93 qapi_WLAN_Raw_Mode_Header_Type_e qapi_WLAN_Raw_Send_Params_t::header_Type

0: Beacon frame, 1: Probe Request frame, 2: QoS data frame, 3: Four addresses data frame, 0xff: Self Defined frame If use Self Defined Header, the frame will be considered as an MGMT frame.>

9.1.4.94 uint16_t qapi_WLAN_Raw_Send_Params_t::seq

Sequence number to be filled in the 802.11 header.

9.1.4.95 uint8_t qapi_WLAN_Raw_Send_Params_t::addr1[__QAPI_WLAN_MAC_LEN]

Address 1.

9.1.4.96 uint8_t qapi_WLAN_Raw_Send_Params_t::addr2[__QAPI_WLAN_MAC_LEN]

Address 2.

9.1.4.97 uint8_t qapi_WLAN_Raw_Send_Params_t::addr3[__QAPI_WLAN_MAC_LEN]

Address 3.

9.1.4.98 uint8_t qapi_WLAN_Raw_Send_Params_t::addr4[__QAPI_WLAN_MAC_LEN]

Address 4. Note that address will not take effect when using self-defined frame

9.1.4.99 uint32_t qapi_WLAN_Raw_Send_Params_t::data_Length

Size of the data to be transmitted.

9.1.4.100 uint8_t* qapi_WLAN_Raw_Send_Params_t::data

Data.

9.1.4.101 uint16_t qapi_WLAN_WPS_Credentials_t::ap_Channel

AP's channel.

9.1.4.102 uint8_t qapi_WLAN_WPS_Credentials_t::ssid[__QAPI_WLAN_MAX_SSID_LEN]

SSID.

9.1.4.103 uint8_t qapi_WLAN_WPS_Credentials_t::ssid_Length

SSID length.

9.1.4.104 qapi_WLAN_Auth_Mode_e qapi_WLAN_WPS_Credentials_t::auth_Mode

Authentication mode.

9.1.4.105 qapi_WLAN_Crypt_Type_e qapi_WLAN_WPS_Credentials_t::encryption_Type

Encryption type.

9.1.4.106 uint8_t qapi_WLAN_WPS_Credentials_t::key_Index

Index of the key.

9.1.4.107 uint8_t qapi_WLAN_WPS_Credentials_t::key[__QAPI_WLAN_WPS_MAX_KEY_LEN+1]

Key.

9.1.4.108 uint8_t qapi_WLAN_WPS_Credentials_t::key_Length

Key length.

9.1.4.109 uint8_t qapi_WLAN_WPS_Credentials_t::mac_Addr[__QAPI_WLAN_MAC_LEN]

MAC address.

9.1.4.110 uint16_t qapi_WLAN_Aggregation_Params_t::tx_TID_Mask

Bitmask to enable Tx A-MPDU aggregation.

9.1.4.111 uint16_t qapi_WLAN_Aggregation_Params_t::rx_TID_Mask

Bitmask to enable Rx A-MPDU aggregation.

9.1.4.112 uint8_t qapi_WPS_Scan_List_Entry_t::ssid[__QAPI_WLAN_MAX_SSID_LEN]

SSID.

9.1.4.113 uint8_t qapi_WPS_Scan_List_Entry_t::macaddress[__QAPI_WLAN_MAC_LEN]

MAC address.

9.1.4.114 uint16_t qapi_WPS_Scan_List_Entry_t::channel

Wireless channel.

9.1.4.115 uint8_t qapi_WPS_Scan_List_Entry_t::ssid_Len

SSID length.

9.1.4.116 qapi_WPS_Scan_List_Entry_t qapi_WLAN_WPS_Start_t::ssid_info

SSID information.

9.1.4.117 uint8_t qapi_WLAN_WPS_Start_t::wps_Mode

WPS pushbutton or WPS PIN.

9.1.4.118 uint8_t qapi_WLAN_WPS_Start_t::timeout_Seconds

WPS timeout in seconds.

9.1.4.119 uint8_t qapi_WLAN_WPS_Start_t::connect_Flag

Connect action (TRUE/FALSE) after intial WPS success.

9.1.4.120 uint8_t qapi_WLAN_WPS_Start_t::pin[__QAPI_WLAN_WPS_PIN_LEN]

PIN.

9.1.4.121 uint8_t qapi_WLAN_WPS_Start_t::pin_Length

PIN length.

9.1.4.122 uint32_t qapi_WLAN_Cipher_t::ucipher

Unicast cipher.

9.1.4.123 uint32_t qapi_WLAN_Cipher_t::mcipher

Multicast cipher.

9.1.4.124 uint16_t qapi_WLAN_Netparams_t::ap_Channel

Wireless channel for the AP.

9.1.4.125 int8_t qapi_WLAN_Netparams_t::ssid[__QAPI_WLAN_MAX_SSID_LENGTH]

[OUT] Network SSID.

9.1.4.126 int16_t qapi_WLAN_Netparams_t::ssid_Len

[OUT] Number of valid chars in ssid[].

9.1.4.127 qapi_WLAN_Cipher_t qapi_WLAN_Netparams_t::cipher

[OUT] Network cipher type values not defined.

9.1.4.128 uint8_t qapi_WLAN_Netparams_t::key_Index

[OUT] For WEP only; key index for Tx.

9.1.4.129 uint8_t { ... } ::wepkey[__QAPI_WLAN_PASSPHRASE_LEN+1]

WEP key.

9.1.4.130 uint8_t { ... } ::passphrase[__QAPI_WLAN_PASSPHRASE_LEN+1]

Passphrase.

9.1.4.131 union { ... } qapi_WLAN_Netparams_t::u

[OUT] Security key or passphrase.

9.1.4.132 uint8_t qapi_WLAN_Netparams_t::sec_Type

[OUT] Security type.

9.1.4.133 uint8_t qapi_WLAN_Netparams_t::error

[OUT] Error code.

9.1.4.134 uint8_t qapi_WLAN_Netparams_t::dont_Block

[IN] 1 – Returns immediately if the operation is not complete

0 – Blocks until the operation completes

9.1.4.135 uint8_t qapi_WLAN_App_Ie_Params_t::mgmt_Frame_Type

Frame in which IE is to be added.

9.1.4.136 uint8_t qapi_WLAN_App_Ie_Params_t::ie_Len

IE length.

9.1.4.137 uint8_t* qapi_WLAN_App_Ie_Params_t::ie_Info

Application specified IE.

9.1.4.138 uint8_t qapi_WLAN_Set_Txpower_Params_t::txpower

9.1.4.139 qapi_WLAN_TX_Power_Policy_e qapi_WLAN_Set_Txpower_Params_t::policy

9.1.4.140 uint16_t qapi_WLAN_Reg_t::start_freq

Start frequency.

9.1.4.141 uint16_t qapi_WLAN_Reg_t::end_freq

End frequency.

9.1.4.142 uint8_t qapi_WLAN_Reg_t::reg_power

Regulatory power.

9.1.4.143 uint8_t qapi_WLAN_Reg_t::ant_gain

Antenna gain.

9.1.4.144 uint16_t qapi_WLAN_Reg_t::flag_info

Flag information.

9.1.4.145 uint16_t qapi_WLAN_Reg_t::max_bw

Maximum bandwidth

9.1.4.146 uint8_t qapi_WLAN_Reg_Evt_t::alpha[3]

Alpha.

9.1.4.147 uint8_t qapi_WLAN_Reg_Evt_t::num_2g_reg_rules

2G regulatory rules.

9.1.4.148 uint8_t qapi_WLAN_Reg_Evt_t::num_5g_reg_rules

5G regulatory rules.

9.1.4.149 qapi_WLAN_Reg_t qapi_WLAN_Reg_Evt_t::reg_rules[QAPI_MAX_REG_RULES]

Data structure used by the application to interpret the regulatory information the device supports.

9.1.4.150 uint8_t qapi_WLAN_Get_Power_Evt_t::reg_power

9.1.4.151 uint8_t qapi_WLAN_Get_Power_Evt_t::ctl_power

Regulatory powe. r

9.1.4.152 uint16_t qapi_WLAN_Get_Power_Evt_t::target_power

Conformance test limit power.

9.1.4.153 uint16_t qapi_WLAN_Get_Power_Evt_t::real_power

The targeted maximum TX power.

9.1.4.154 uint8_t qapi_WLAN_Set_Rate_Params_t::ra_ON

Flag indicating whether automatic rate adaptation is enabled. Use NT_RA_OFF to disable or NT_RA_ON to enable.

9.1.4.155 uint8_t qapi_WLAN_Set_Rate_Params_t::rate_staid

Station ID for which the rate configuration applies.

9.1.4.156 uint8_t qapi_WLAN_Set_Rate_Params_t::rate_p_rate

Primary transmission rate. If transmission fails NT_DEVCFG_RETRY_THRESHOLD0 times (configurable; see rate_adaptation_debug.xml), the secondary rate will be attempted.

9.1.4.157 uint8_t qapi_WLAN_Set_Rate_Params_t::rate_s_rate

Secondary transmission rate. If transmission fails an additional (NT_DEVCFG_RETRY_THRESHOLD1 - NT_DEVCFG_RETRY_THRESHOLD0) times, the tertiary rate will be attempted.

9.1.4.158 uint8_t qapi_WLAN_Set_Rate_Params_t::rate_t_rate

9.1.4.159 uint32_t qapi_WLAN_Listen_Interval_Params_t::time

Listen interval in Time Units (TU), where 1 TU = 1024 microseconds.

9.1.4.160 uint32_t qapi_WLAN_Listen_Interval_Params_t::round_type

Determines how the beacon interval is rounded when calculating (time / beacon interval).

9.1.4.161 uint8_t qapi_WLAN_Edca_Params_t::qid

Queue ID∗./ uint8_t aifsn; /∗∗< Arbitration inter frame spacing number.

9.1.4.162 uint16_t qapi_WLAN_Edca_Params_t::cw_min

Minimum value of contention window.

9.1.4.163 uint16_t qapi_WLAN_Edca_Params_t::cw_max

Maximum value of contention window.

9.1.4.164 uint16_t qapi_WLAN_Edca_Params_t::txop_limit

Transmission opportunity limit.

9.1.4.165 uint16_t qapi_WLAN_BA_Window_Params_t::ack_timeout

BA window ack timeout in us.

9.1.4.166 uint16_t qapi_WLAN_BA_Window_Params_t::delay

Propagation time in numbers of SM clock cycles.

9.1.4.167 uint16_t qapi_WLAN_BA_Window_Size_t::tx_size

TX BA window size.

9.1.4.168 uint16_t qapi_WLAN_BA_Window_Size_t::rx_size

RX BA window size.

9.2 WLAN errors

9.2.1 Define Documentation

9.2.1.1 #define QAPI_WLAN_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 1)))

Generic error.

9.2.1.2 #define QAPI_WLAN_ERR_DEVICE_NOT_FOUND ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 3)))

API was not able to find a PCI device.

9.2.1.3 #define QAPI_WLAN_ERR_NO_MEMORY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 4)))

API was not able to allocate memory dynamically.

9.2.1.4 #define QAPI_WLAN_ERR_MEMORY_NOT_AVAIL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 5)))

Memory region is not free for mapping.

9.2.1.5 #define QAPI_WLAN_ERR_NO_FREE_DESC ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 6)))

Free descriptors are not available.

9.2.1.6 #define QAPI_WLAN_ERR_BAD_ADDRESS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 7)))

Address does not match the descriptor.

9.2.1.7 #define QAPI_WLAN_ERR_WIN_DRIVER_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 8)))

Used in NT_HW version if a problem occures at initialization.

9.2.1.8 #define QAPI_WLAN_ERR_REGS_NOT_MAPPED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 9)))

Registers are not correctly mapped.

9.2.1.9 #define QAPI_WLAN_ERR_EPERM ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 10)))

Operation is not permitted.

9.2.1.10 #define QAPI_WLAN_ERR_EACCES ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 11)))

Access is denied.

9.2.1.11 #define QAPI_WLAN_ERR_ENOENT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 12)))

No such entry; search failed.

9.2.1.12 #define QAPI_WLAN_ERR_EEXIST ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 13)))

The object already exists.

9.2.1.13 #define QAPI_WLAN_ERR_EFAULT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 14)))

Bad address error.

9.2.1.14 #define QAPI_WLAN_ERR_EBUSY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 15)))

Object is busy.

9.2.1.15 #define QAPI_WLAN_ERR_EINVAL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 16)))

Invalid parameter.

9.2.1.16 #define QAPI_WLAN_ERR_EMSGSIZE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 17)))

Inappropriate message buffer length.

9.2.1.17 #define QAPI_WLAN_ERR_ECANCELED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 18)))

Operation was canceled.

9.2.1.18 #define QAPI_WLAN_ERR_ENOTSUP ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 19)))

Operation is not supported.

9.2.1.19 #define QAPI_WLAN_ERR_ECOMM ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 20)))

Communication error on send.

9.2.1.20 #define QAPI_WLAN_ERR_EPROTO ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 21)))

Protocol error.

9.2.1.21 #define QAPI_WLAN_ERR_ENODEV ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 22)))

Incorrect device.

9.2.1.22 #define QAPI_WLAN_ERR_EDEVNOTUP ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 23)))

Device is not UP.

9.2.1.23 #define QAPI_WLAN_ERR_NO_RESOURCE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 24)))

No resources available for the requested operation.

9.2.1.24 #define QAPI_WLAN_ERR_HARDWARE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 25)))

Hardware failure

9.2.1.25 #define QAPI_WLAN_ERR_PENDING ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 26)))

Asynchronous routine. Firmware will send results later, typically in callback.

9.2.1.26 #define QAPI_WLAN_ERR_EBADCHANNEL ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 27)))

The specified channel cannot be used.

9.2.1.27 #define QAPI_WLAN_ERR_DECRYPT_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 28)))

Decryption error.

9.2.1.28 #define QAPI_WLAN_ERR_PHY_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 29)))

Receiver error at the physical layer.

9.2.1.29 #define QAPI_WLAN_ERR_CONSUMED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 30)))

Object was consumed.

9.2.1.30 #define QAPI_WLAN_ERR_CLONE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 31)))

The buffer is cloned.

9.2.1.31 #define QAPI_WLAN_ERR_HW_CONFIG_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 32)))

Error in a GPIO operation.

9.2.1.32 #define QAPI_WLAN_ERR_SOCKCXT_NOT_FOUND ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 33)))

Socket context was not found.

9.2.1.33 #define QAPI_WLAN_ERR_UNKNOWN_CMD ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 34)))

Unknown socket command.

9.2.1.34 #define QAPI_WLAN_ERR_SOCK_UNAVAILABLE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 35)))

Socket limit was reached.

9.2.1.35 #define QAPI_WLAN_ERR_MEMFREE_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 36)))

Error while freeing a resource.

9.2.1.36 #define QAPI_WLAN_ERR_BUS_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 37)))

Error while reading/writing bus.

9.2.1.37 #define QAPI_WLAN_ERR_QOSAL_INVALID_TASK_ID ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 38)))

Invalid task ID.

9.2.1.38 #define QAPI_WLAN_ERR_QOSAL_INVALID_PARAMETER ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 39)))

Invalid parameter.

9.2.1.39 #define QAPI_WLAN_ERR_QOSAL_INVALID_POINTER ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 40)))

Invalid pointer.

9.2.1.40 #define QAPI_WLAN_ERR_QOSAL_ALREADY_EXISTS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 41)))

Object already exists.

9.2.1.41 #define QAPI_WLAN_ERR_QOSAL_INVALID_EVENT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 42)))

Invalid event.

9.2.1.42 #define QAPI_WLAN_ERR_QOSAL_EVENT_TIMEOUT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 43)))

Event timed out.

9.2.1.43 #define QAPI_WLAN_ERR_QOSAL_INVALID_MUTEX ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 44)))

Incorrect mutex.

9.2.1.44 #define QAPI_WLAN_ERR_QOSAL_TASK_ALREADY_LOCKED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 45)))

Specified task is locked by another thread.

9.2.1.45 #define QAPI_WLAN_ERR_QOSAL_MUTEX_ALREADY_LOCKED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 46)))

Specified mutex is locked by another thread.

9.2.1.46 #define QAPI_WLAN_ERR_QOSAL_OUT_OF_MEMORY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 47)))

No memory is available for the operation.

9.2.1.47 #define QAPI_WLAN_ERR_IMG_VERIFY_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 48)))

WLAN image fails verification.

9.2.1.48 #define QAPI_WLAN_ERR_FLASH_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_WIFI, 49)))

Fails to open or read flash.

9.4 WLAN parameter group information

9.4.1 Define Documentation

9.4.1.1 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS 1

Macro that indicates the group ID that can be used to configure wireless parameters of the WLAN subsystem.

9.4.1.2 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SECURITY 2

Macro that indicates the group ID that can be used to configure security parameters of the WLAN subsystem.

9.4.1.3 #define __QAPI_WLAN_PARAM_GROUP_P2P 3

Macro that indicates the group ID that can be used to configure various P2P (Peer-to-Peer/Wi-Fi Direct) parameters.

9.4.1.4 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE 1

Command ID to set/get the operating mode of a given virtual device in the WLAN subsystem.

Mode-specific parameters should be set only after setting the mode (AP/STA) using this command.

If a concurrent mode of operation is enabled, the SAP can only be operated in virtual device 0, and the STA mode of operation can only operate in virtual device 1.

In the case of a single device mode of operation, device 0 can be used for both STA and SAP mode of operation.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

`in,out`	*opMode*	Address of the variable type qapi_WLAN_Dev_Mode_e

See also

qapi_WLAN_Dev_Mode_e

9.4.1.5 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_CHANNEL 2

Command ID to set/get the operating wireless channel of a given virtual device in the WLAN subsystem.

For set/get operations, channel values are set in numbers (channel number 1-14, 36-165) and not in frequency values.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

`in,out`	*uint32_t*	Variable that holds the channel number.

9.4.1.6 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_TX_POWER_IN_DBM 4

Command ID to set/get the transmit power in dBm of a given virtual device.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*uint32_t*	Address of the variable that holds the power value.

9.4.1.7 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SSID 5

Command ID to set/get the SSID of/for a given virtual device in the WLAN subsystem.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*uint8_t[ ]*	Unsigned byte array of size __QAPI_WLAN_MAX_SSID_LENGTH.

See also

__QAPI_WLAN_MAX_SSID_LENGTH

9.4.1.8 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_BSSID 6

Command ID to set/get the BSSID of/for a given virtual device in the WLAN subsystem when operation in STA mode.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*uint8_t[ ]*	Unsigned byte array of size __QAPI_WLAN_MAC_LEN.

See also

__QAPI_WLAN_MAC_LEN

9.4.1.9 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_PHY_MODE 7

Command ID to set/get the wireless PHY mode of/for a given virtual device.

The Set operation for this should be done before establishing a connection.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_Phy_-Mode_e*	Required PHY mode should be specified.

See also

qapi_WLAN_Phy_Mode_e

9.4.1.10 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ALLOW_TX_RX_AGGR_SET_TID 9

Command ID to allow/disallow aggregation for Tx and Rx on a TID basis.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_Aggregation_Params_t*	Structure populated with required aggregation parameters for Tx and Rx.

See also

qapi_WLAN_Aggregation_Params_t

9.4.1.11 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_POWER_MODE_PARAMS 14

Command ID to set/get the virtual device power mode.

The supported modes are Power Save mode (also known as REC_POWER) and Performance mode (MAX_PERF). Applications are recommended to configure virtual devices in Performance mode when more than one virtual device is connected (concurrency enabled).

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_Power_Mode_Params_t*	Power mode configurations.

See also

qapi_WLAN_Power_Mode_Params_t

9.4.1.12 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_LISTEN_INTERVAL_IN_TU 32

Command ID to configure the 802.11 listen interval when operating in Station mode. This value will be used in the listen interval field of the association request frame.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Listen interval in multiples of beacon intervals (a value of 1 corresponds to 1 beacon interval).

9.4.1.13 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RSSI 33

Command ID to get the RSSI of the associated peer.

NOTE This parameter can only be used with qapi_WLAN_Get_Param().

Parameters

out	*uint8_t*	RSSI value variable received from the firmware.

9.4.1.14 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AMSDU_RX 39

Command ID to set reiceive AMSDU enable or disable.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Set to TRUE to enable AMSDU receive mode, FALSE otherwise.

9.4.1.15 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_BEACON_INTERVAL_IN_TU 40

Command ID to set the beacon interval (in time units) when operating in SoftAP mode. One TU = 1024 microseconds.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Number of TUs between every beacon in SoftAP mode.

9.4.1.16 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_ENABLE_HIDDEN_MODE 41

Command ID to enable/disable the hidden SSID feature when operating a virtual device in SoftAP mode.

This should be done after setting the operating mode as AP and before committing the AP profile using qapi_WLAN_Commit().

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Set to TRUE to enable the hidden SSID feature, FALSE otherwise.

Dependencies

Should be set after setting the operating mode to AP by issuing __QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE.
Should be set before invoking qapi_WLAN_Commit() to start SoftAP.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE
qapi_WLAN_Commit()

9.4.1.17 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_INACTIVITY_TIME-_IN_MINS 43

Command ID to set an AP's inactivity period in minutes.

If no keepalive frames are received from an associated station during this period, the AP deassociates that station.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Inactivity interval for associated stations in minutes.

9.4.1.18 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_DTIM_INTERVA-L 45

Command ID to change the DTIM interval when operating a virtual device in SoftAP mode.

This setting should be done before committing the AP profile using qapi_WLAN_Commit().

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	DTIM interval in multiples of the beacon interval.

Dependencies

Should be set after setting the operating mode to AP by issuing __QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE.
Should be set before invoking qapi_WLAN_Commit() to start SoftAP.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE
qapi_WLAN_Commit

9.4.1.19 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_11N_HT 60

Command ID to set wireless 11n HT parameters of a given virtual device. The set operation for this should be done before establishing a connection.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_11n_H-T_Config_s*	Required 11n HT configuration must be specified.

See also

qapi_WLAN_11n_HT_Config_s

9.4.1.20 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_BMISS_CONFIG 61

Command ID to set Beacon Miss configuration.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in,out	*qapi_WLAN_Sta_Config_Bmiss_Config_t*	Beacon Miss parameters to be set.

See also

qapi_WLAN_Sta_Config_Bmiss_Config_t

9.4.1.21 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_CONCURRENCY_MODE 82

Command ID to set/get the concurrency mode of device in the WLAN subsystem.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in,out	*concurrency*	mode Address of the variable type qapi_WLAN_DEV_Mode_e

See also

qapi_WLAN_DEV_Mode_e

9.4.1.22 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RTS 84

Command ID to enable/disable RTS/CTS protection when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*uint32_t*	Set 1 to enable RTS/CTS protection, 0 to be disabled.

9.4.1.23 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RTS_RATE_2G 85

Command ID to fix RTS rate in 2G when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*uint32_t*	0: 1Mbps 1: 6Mbps

9.4.1.24 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCA_PARAM 86

Command ID to adjust edca parameters when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*qapi_WLAN_Edca_Params_t*	Set edca parameters for queue 0-7.

See also

qapi_WLAN_Edca_Params_t

9.4.1.25 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_PER_UPPER_THRESHOLD 87

Command ID to adjust PER upper threshold when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*uint32_t*	Set PER upper threshold to 0-100.

9.4.1.26 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_BA_WINDOW 88

Command ID to adjust BA window size when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*qapi_WLAN_BA_Window_Params_t*	Set BA window size.

See also

qapi_WLAN_BA_Window_Params_t

9.4.1.27 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SLOT_TIME 89

Command ID to adjust BA window size when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Parameters

in	*uint32_t*	change slot time to 9us/20us.

9.4.1.28 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCCA_THRESHOLD 90

Command ID to adjust EDCCA threshold when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Get_Param().

Set EDCCA threshold to 0-100.

9.4.1.29 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RSP_RATE 91

Command ID to set rsp rate in Station mode. The set operation for this should be done after establishing a connection.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint8_t*	RspRate idx, only support 8:11g 6Mbps 16:11n 6.5Mbps.

9.4.1.30 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_BA_WINDOW_SIZE 92

Command ID to adjust BA window size when operating in Station mode.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_BA_Window_Size_t*	BA window size.

9.4.1.31 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_PROTECTION_MODE 93

Command ID to set protection mode when operating in Station mode.

NOTE This parameter can be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Set 1 to enable CTS_TO_SELF protection, 0 to be disabled.

9.4.1.32 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_ENCRYPTION_TYPE 1

Command ID to set/get the encryption mode for an upcoming association operation.

NOTE This parameter can be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_Crypt_Type_e*	Encryption mode to be set.

Dependencies

Encryption mode must be set before connecting to the peer.

See also

qapi_WLAN_Crypt_Type_e

9.4.1.33 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PASSPHRASE 3

Command ID to set the passphrase for the upcoming WPA/WPA2 association procedure.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint8_t[ ]*	Passphrase to be set. The passphrase length should not exceed __QAPI_WLAN_PASSPHRASE_LEN.

Dependencies

This should be done before initiating an association.

See also

__QAPI_WLAN_PASSPHRASE_LEN

9.4.1.34 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WPS_CREDENTIALS 6

Command ID to set WPS credentials received after WPS negotiation with the peer. These are the credentials that will be used for secure association with the peer.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_WPS_-Credentials_t*	WPS credential information to be used for a secure association.

Dependencies

This should be done after WPS negotiation is completed and before performing a secure association.

See also

qapi_WLAN_WPS_Credentials_t

9.4.1.35 #define __QAPI_WLAN_PARAM_GROUP_P2P_CONFIG_PARAMS 0

Command ID to configure P2P device parameters. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Config_Params_t*	Device parameters to be configured.

Dependencies

P2P mode must be enabled before device parameter configuration.

See also

#qapi_WLAN_P2P_Config_Params_t
qapi_WLAN_Set_Param()
qapi_WLAN_P2P_Enable()

9.4.1.36 #define __QAPI_WLAN_PARAM_GROUP_P2P_OPPS_PARAMS 1

Command ID to configure P2P opportunistic power save parameters. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Opps_Params_t*	Opportunistic power save parameters.

Dependencies

Opportunistic power save mode applies only to P2P group owners. A device can initiate an autonomous group owner operation or it can become a group owner through a group owner negotiation process.

See also

qapi_WLAN_P2P_Opps_Params_t
qapi_WLAN_Set_Param()
qapi_WLAN_P2P_Connect()
qapi_WLAN_P2P_Start_Go()

9.4.1.37 #define __QAPI_WLAN_PARAM_GROUP_P2P_NOA_PARAMS 2

Command ID to configure P2P notice of absence (NOA) parameters. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Noa_Params_t*	Notice of absence configuration parameters.

Dependencies

NOA applies only to P2P group owners. A device can initiate autonomous group owner operation or it can become a group owner through a group owner negotiation process.

See also

qapi_WLAN_P2P_Noa_Params_t
qapi_WLAN_Set_Param()
qapi_WLAN_P2P_Connect()
qapi_WLAN_P2P_Start_Go()

9.4.1.38 #define __QAPI_WLAN_PARAM_GROUP_P2P_NODE_LIST 3

Command ID to get a list of P2P peer devices (/nodes) found through the P2P find phase. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_P2P_-Node_List_Params_t*	List to get peer device information.

Dependencies

Nearby devices must be discovered using qapi_WLAN_P2P_Find() before the node list can show those.

See also

qapi_WLAN_P2P_Node_List_Params_t qapi_WLAN_Get_Param()
qapi_WLAN_P2P_Find()

9.4.1.39 #define __QAPI_WLAN_PARAM_GROUP_P2P_NETWORK_LIST 4

Command ID to get a list of P2P groups stored in the nonvolatile storage. This list is persistent across reboots. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_P2P_-Network_List_-Params_t*	List to get persistent connections information.

Dependencies

Persistent groups should be formed by using the 'persistent' option while connecting to a peer device and/or authenticating a peer device for a connection. Without this option, the connections will be lost when the device reboots.

See also

qapi_WLAN_P2P_Network_List_Params_t
qapi_WLAN_Get_Param()
qapi_WLAN_P2P_Connect()
qapi_WLAN_P2P_Auth()

9.4.1.40 #define __QAPI_WLAN_PARAM_GROUP_P2P_LISTEN_CHANNEL 6

Command ID to configure a device listen channel that is used for the device to be discoverable.

The listen channel should be one of the social channels (1, 6, 11 for 2.4 Ghz) and it should remain the same until the device discovery completes. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'listen_Channel' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Listen_Channel_t*	Listen channel information.

See also

qapi_WLAN_P2P_Listen_Channel_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.41 #define __QAPI_WLAN_PARAM_GROUP_P2P_SSID_POSTFIX 8

Command to configure the postfix to be appended to the P2P group SSID.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_-Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'ssid_Postfix' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Ssid_Postfix_t*	SSID postfix information.

Dependencies

Only a group owner can add an SSID postfix.

See also

qapi_WLAN_P2P_Set_Ssid_Postfix_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.42 #define __QAPI_WLAN_PARAM_GROUP_P2P_INTRA_BSS 9

Command ID to enable/disable intra BSS data forwarding support.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_-Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'intra_Bss' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Intra_Bss_t*	Flag to enable/disable intra BSS data forwarding support.

See also

qapi_WLAN_P2P_Set_Intra_Bss_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.43 #define __QAPI_WLAN_PARAM_GROUP_P2P_GO_INTENT 11

Command ID to configure a device's group owner intent.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_-Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'go_Intent' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Go_Intent_t*	Device's group owner intent.

See also

qapi_WLAN_P2P_Set_Go_Intent_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.44 #define __QAPI_WLAN_PARAM_GROUP_P2P_DEV_NAME 12

Command ID to configure a device name.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_-Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'device_Name' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Dev_Name_t*	Device name to be configured.

See also

qapi_WLAN_P2P_Set_Dev_Name_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.45 #define __QAPI_WLAN_PARAM_GROUP_P2P_OP_MODE 13

Command ID to configure a device's P2P operating mode.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param() to set the device operation. It is used only when the mode to be set is __QAPI_WLAN_P2P_CLIENT before mode to client before calling qapi_WLAN_P2P_Join(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'mode' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Mode_t*	Operating mode.

See also

__QAPI_WLAN_P2P_CLIENT
qapi_WLAN_P2P_Set_Mode_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.46 #define __QAPI_WLAN_PARAM_GROUP_P2P_CCK_RATES 14

Command ID to enable/disable a complementary code keying (CCK) modulation scheme.

Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_-Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'cck_Rates' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Cck_Rates_t*	Enable/disable a CCK modulation scheme.

See also

qapi_WLAN_P2P_Set_Cck_Rates_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.47 #define __QAPI_WLAN_PARAM_GROUP_P2P_DISCOVERABLE_INTERVA-L 15

Command ID to set P2P discoverable interval. Use this parameter with group ID __QAPI_WLAN_PARAM_GROUP_P2P to call qapi_WLAN_Set_Param(). Alternately, applications can use an object of structure qapi_WLAN_P2P_Set_Cmd_t with this macro as 'config_Id' and 'discoverable_Interval' members of union 'val' as data to call qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_P2P_-Set_Discoverable_-Interval_t*	Set P2P discoverable interval in milliseconds.

See also

qapi_WLAN_P2P_Set_Discoverable_Interval_t
qapi_WLAN_P2P_Set_Cmd_t
qapi_WLAN_Set_Param()

9.4.1.48 #define __QAPI_WLAN_PARAM_GROUP_P2P_GO_PARAMS 5

Command ID used for calling qapi_WLAN_Set_Param() to configure P2P group owner parameters.

10 Fatal Error Manager

10.1 Fatal error

Fatal Error Manager (FEM)

Complex software systems often run into unrecoverable error scenarios. These fatal errors cause the system to abruptly abort execution, since there is no recovery path. By nature, fatal errors are difficult to debug because detailed information related to the error is not preserved. The fatal error manager (FEM) service provides its clients a way to handle unrecoverable errors in a graceful, debug-friendly fashion. It exposes a macro which, when called after a catastrophic error, preserves pertinent information to aid in debug before resetting the system.

```
{.c}

   * The following code snippet demonstrates the use of this interface. The
     example
   * dynamically allocates a region of memory, failing in which it
   * asserts the code. This macro populates the debug information in a global
   * variable 'coredump' with line number, file name, and user parameters.
   * It also dumps the contents of general purpose registers and invokes
   * various user callbacks before resetting the system. The header file
   * qapi_fatal_err.h should be included before calling the macro.

  char * c;

  c = malloc(sizeof(char));
  if ( c == NULL )
  {
    QAPI_FATAL_ERR(0,0,0);
  }
```

10.1.1 Define Documentation

10.1.1.1 #define QAPI_FATAL_ERR(*param1, param2, param3*)

Fatal error handler macro.

This function allows for graceful handling of fatal errors. It preserves information related to fatal crashes at a well-known location (typically a global variable "coredump"). Preserved information captures the source module name and line number, user-provided values, and contents of general purpose registers used by the underlying CPU architecture. After invoking several notification callbacks, it resets the system.

Parameters

in	*param1*	User-provided parameter to be logged in coredump.
in	*param2*	User-provided parameter to be logged in coredump.
in	*param3*	User-provided parameter to be logged in coredump.

NOTE This macro does not return. It should only be used to gracefully handle unrecoverable errors and restart the system.

10.1.2 Data Structure Documentation

10.1.2.1 struct qapi_Err_const_t

Debug information structure.

This structure is used to capture the module name and line number in the source file where a fatal error was detected. Reference to an instance of this structure is passed as a parameter to the qapi_err_fatal_internal() function.

Data fields

Type	Parameter	Description
const char *	fname	Pointer to the source file name.
uint16_t	line	Line number in the source module.

10.1.3 Function Documentation

10.1.3.1 void qapi_err_fatal_internal (const qapi_Err_const_t * *err_const,* uint32_t *param1,* uint32_t *param2,* uint32_t *param3*)

Fatal error handler.

This function implements back-end functionality supported by macro QAPI_FATAL_ERR. It preserves debug information at a well-known location (typically a global variable "coredump"). Preserved information captures the source module name and line number, user-provided values, and contents of general purpose registers for underlying CPU architecture. After invoking several notification callbacks, it resets the system.

Parameters

in	*err_const*	Reference to the structure record line number and module name.
in	*param1*	Client-provided parameter saved with debug information.
in	*param2*	Client-provided parameter saved with debug information.
in	*param3*	Client-provided parameter saved with debug information.

Note

This function does not return. It should only be used to gracefully handle unrecoverable errors and restart the system. Clients should not call the function directly. Instead, they should use the macro QAPI_FATAL_ERR to access the functionality to ensure that all relevant debug information is carried forward.

10.1.4 Variable Documentation

10.1.4.1 const char* qapi_Err_const_t::fname

Pointer to the source file name.

10.1.4.2 uint16_t qapi_Err_const_t::line

Line number in the source module.

11 GPIO module

11.1 General Purpose Input/Output interface (GPIO)

The GPIO module provides access to general-purpose input/output (GPIO) pins the value of which consist of one of two voltage settings (high or low) and the behavior of which can be programmed using software.

Typical usage:

- qapi_GPIO_Config() – Config GPIO parameters, include function, direction, pull-type, and drive strength.
- qapi_GPIO_Set() – Set the GPIO output value.
- qapi_GPIO_Get() – Get the GPIO input value.
- qapi_GPIO_Enable_Interrupt() – Register and enable GPIO input interrupt.
- qapi_GPIO_Disable_Interrupt() – Unregister the GPIO interrupt function.
- qapi_GPIO_Mux_Pin_Get() Get the Mux function pin name.

11.1.1 Data Structure Documentation

11.1.1.1 struct qapi_GPIO_Alt_Config_s

GPIO alternative configuration.

This structure is used to specify the alternative configurations of GPIOs for different hardware designs.

Data fields

Type	Parameter	Description
uint16_t	PIOFunc: 4	PIO function select.
uint16_t	Dir: 1	Direction (input or output).
uint16_t	Pull: 2	Pull value.
uint16_t	Drive: 3	Drive strength.
uint16_t	se_port: 6	PIO function select.

11.1.1.2 struct qapi_GPIO_Config_s

GPIO configuration.

This structure is used to specify the configuration for a GPIO on the SoC. The GPIO can be configured as an input or output that can be driven high or low by the software. The interface also allows the SoC PIOs to be configured for alternate functionality.

Data fields

Type	Parameter	Description
qapi_GPIO_-Direction_t	Dir	Direction (input or output).
qapi_GPIO_-Pull_t	Pull	Pull value.
qapi_GPIO_-Drive_t	Drive	Drive strength.

11.1.1.3 struct qapi_GPIO_CB_List_s

GPIO interrupt callback list structure.

This structure is used to store the GPIO pin interrupt callback function and parameter in a linked list.

Data fields

Type	Parameter	Description
qapi_GPIO_Id-_t	GPIO_ID	GPIO pin number.
qapi_GPIO_C-B_t	Func	GPIO callback function.
qapi_GPIO_C-B_Data_t	Data	GPIO callback parameter.
struct qapi_GP-IO_CB_List_s *	Next	Pointer to the next GPIO in the linked list callback structure.

11.1.2 Typedef Documentation

11.1.2.1 typedef struct qapi_GPIO_Alt_Config_s qapi_GPIO_Alt_Config_t

GPIO alternative configuration.

This structure is used to specify the alternative configurations of GPIOs for different hardware designs.

11.1.2.2 typedef struct qapi_GPIO_Config_s qapi_GPIO_Config_t

GPIO configuration.

This structure is used to specify the configuration for a GPIO on the SoC. The GPIO can be configured as an input or output that can be driven high or low by the software. The interface also allows the SoC PIOs to be configured for alternate functionality.

11.1.2.3 typedef uint32_t qapi_GPIO_CB_Data_t

GPIO interrupt callback data type.

This is the data type of the argument passed into the callback that is registered with the GPIO interrupt module. The value to pass is given by the client at registration time.

11.1.2.4 typedef void(∗ qapi_GPIO_CB_t)(qapi_GPIO_CB_Data_t Data)

GPIO interrupt callback function definition.

GPIO interrupt clients pass a function pointer of this format into the registration API.

Parameters

in	*Data*	Callback data.

11.1.2.5 typedef struct qapi_GPIO_CB_List_s qapi_GPIO_CB_List_t

GPIO interrupt callback list structure.

This structure is used to store the GPIO pin interrupt callback function and parameter in a linked list.

11.1.3 Enumeration Type Documentation

11.1.3.1 enum qapi_GPIO_Id_t

GPIO pin number.

Enumerator:

QAPI_GPIO_ID0_E GPIO 0.
QAPI_GPIO_ID1_E GPIO 1.
QAPI_GPIO_ID2_E GPIO 2.
QAPI_GPIO_ID3_E GPIO 3.
QAPI_GPIO_ID4_E GPIO 4.
QAPI_GPIO_ID5_E GPIO 5.
QAPI_GPIO_ID6_E GPIO 6.
QAPI_GPIO_ID7_E GPIO 7.
QAPI_GPIO_ID8_E GPIO 8.
QAPI_GPIO_ID9_E GPIO 9.
QAPI_GPIO_ID10_E GPIO 10.
QAPI_GPIO_ID11_E GPIO 11.
QAPI_GPIO_ID12_E GPIO 12.
QAPI_GPIO_ID13_E GPIO 13.
QAPI_GPIO_ID14_E GPIO 14.
QAPI_GPIO_MAX_ID_E Maximum GPIO.

11.1.3.2 enum qapi_GPIO_Direction_t

GPIO pin direction.

Enumerator:

QAPI_GPIO_INPUT_E Specify the PIO as an INPUT to the SoC.
QAPI_GPIO_OUTPUT_E Specify the PIO as an OUTPUT from the SoC.

11.1.3.3 enum qapi_GPIO_Pull_t

GPIO pin pull type.

Enumerator:

QAPI_GPIO_NO_PULL_E Specify no pull. {Input + NO PULL} is equal to High-Z state.
QAPI_GPIO_PULL_DOWN_E Pull the GPIO down.
QAPI_GPIO_PULL_UP_E Pull the GPIO up.

11.1.3.4 enum qapi_GPIO_Drive_t

GPIO pin drive strength.

Enumerator:

QAPI_GPIO_DRIVE_LOW_E Specify a fast 2 mA drive.
QAPI_GPIO_DRIVE_HIGH_E Specify a fast 4 mA drive.

11.1.3.5 enum qapi_GPIO_Value_t

GPIO output state specification.

Enumerator:

QAPI_GPIO_LOW_VALUE_E Drive the output LOW.
QAPI_GPIO_HIGH_VALUE_E Drive the output HIGH.

11.1.3.6 enum qapi_GPIO_Trigger_t

GPIO interrupt trigger type enumeration for supported triggers.

Enumerator:

QAPI_GPIO_TRIGGER_LEVEL_HIGH_E Level triggered active high.
QAPI_GPIO_TRIGGER_LEVEL_LOW_E Level triggered active low.
QAPI_GPIO_TRIGGER_EDGE_RISING_E Rising-edge triggered.
QAPI_GPIO_TRIGGER_EDGE_FALLING_E Falling-edge triggered.

11.1.4 Function Documentation

11.1.4.1 qapi_Status_t qapi_GPIO_Config (qapi_GPIO_Id_t *GPIO_ID,* qapi_GPIO_Config_t * *Config*)

Changes the SoC PIO configuration.

This function configures an SoC PIO based on a set of fields specified in the configuration structure reference passed in as a parameter.

Parameters

in	*GPIO_ID*	GPIO number.
in	*Config*	PIO configuration to use.

Returns

QAPI_OK – On success.
Error code – On failure.

11.1.4.2 qapi_Status_t qapi_GPIO_Set (qapi_GPIO_Id_t *GPIO_ID,* qapi_GPIO_Value_t *Value*)

Sets the state of an SoC PIO configured as an output GPIO.

This function drives the output of an SoC PIO that has been configured as a generic output GPIO to a specified value.

Parameters

in	*GPIO_ID*	GPIO number.
in	*Value*	Output value.

Returns

QAPI_OK – On success.
Error code – On failure.

11.1.4.3 qapi_Status_t qapi_GPIO_Get (qapi_GPIO_Id_t *GPIO_ID,* qapi_GPIO_Value_t ∗ *Value*)

Reads the state of an SoC PIO.

Parameters

in	*GPIO_ID*	GPIO number.
out	*Value*	Input value.

Returns

QAPI_OK – On success.
Error code – On failure.

11.1.4.4 qapi_Status_t qapi_GPIO_Enable_Interrupt (qapi_GPIO_Id_t *GPIO_ID,* qapi_GPIO_Trigger_t *Trigger,* qapi_GPIO_CB_t *Callback,* qapi_GPIO_CB_Data_t *Data*)

Registers a callback for a GPIO interrupt.

Registers a callback function with the GPIO interrupt controller, and enables the interrupt. This function configures and routes the interrupt accordingly, as well as enabling it in the underlying layers.

Parameters

in	*GPIO_ID*	GPIO number.
in	*Trigger*	Trigger type for the interrupt.
in	*Callback*	Callback function pointer.
in	*Data*	Callback data.

Returns

QAPI_OK – On success.

Error code – On failure.

11.1.4.5 qapi_Status_t qapi_GPIO_Disable_Interrupt (qapi_GPIO_Id_t *GPIO_ID*)

Deregisters a callback for a GPIO interrupt.

Deregisters a callback function from the GPIO interrupt controller, and disables the interrupt. This function deconfigures the interrupt accordingly, and disables it in the underlying layers.

Parameters

in	*GPIO_ID*	GPIO number.

Returns

QAPI_OK – On success.
Error code – On failure.

11.1.4.6 QAPI_GPIO_MUX_PIN_t qapi_GPIO_Mux_Pin_Get (qapi_GPIO_Id_t *GPIO_ID*)

retrive the GPIO MUX configuration.

Parameters

in	*GPIO*	ID.

Returns

return the GPIO Mux Pin defination or error type.

12 HFC APIs

12.1 HFC APIs

12.1.1 Function Documentation

12.1.1.1 qapi_Status_t qapi_hfc_sendto_host_data_pkt (void * *p_buff,* uint8_t * *payload,* uint16_t *len,* uint16_t *info*)

Send data packets to the host.

Parameters

in	*p_buff*	Pointer of buffer.
in	*payload*	Pointer of payload.
in	*len*	Length of payload.
in	*info*	Extra information.

Returns

- QAPI_OK – On success.
- Error code – On failure.

12.1.1.2 qapi_Status_t qapi_hfc_recvfrom_host_data_pkt (void * *p_buff,* uint16_t * *buf_len,* uint32_t *timeout,* uint16_t * *data_len,* uint16_t * *info*)

Receives data packets from the host.

Parameters

in	*p_buff*	Pointer of buffer.
in	*buf_len*	Pointer of buffer length.
in	*timeout*	Timeout in millisecond.
out	*data_len*	Pointer of data length.
out	*info*	Pointer of extra information.

Returns

- QAPI_OK – On success.
- Error code – On failure.

12.1.1.3 qbool_t qapi_hfc_sendto_host_config_pkt (uint32_t * *p_buf,* uint16_t *len*)

Sends configuration packets to the host.

Parameters

in	*p_buf*	Pointer to the buffer of the configuration packet.
in	*len*	Length of the configuration packet.

Returns

- TRUE – On success.
- FALSE – On failure.

12.1.1.4 qapi_Status_t qapi_hfc_recvfrom_host_msg (hfc_msg_t * *msg,* uint32_t *timeout*)

Receives data packets from the host.

Parameters

in	*msg*	Message pointer.
in	*timeout*	Timeout in milliseconds.

Returns

- QAPI_OK – On success.
- Error code – On failure.

12.1.1.5 qapi_Status_t qapi_hfc_set_gpio_assert_info (f2a_event_type *event*)

Sets the WLAN state.

Parameters

in	*event*	

Returns

- QAPI_OK – On success.
- QAPI_ERROR – On failure.

13 HTTP Client

13.1 HTTP client

13.1.1 Define Documentation

13.1.1.1 #define QAPI_NET_HTTPC_CHUNKED_MASK 0x80

HTTP client chunk encoded.

13.1.1.2 #define QAPI_NET_HTTPC_WITH_HEADER_MASK 0x01

HTTP client with header mask.

13.1.2 Data Structure Documentation

13.1.2.1 struct qapi_Ssl_Config

SSL configuration.

Data fields

Type	Parameter	Description
uint16_t	protocol	
int	force_-ciphersuite[2]	
char *	server_name	
char *	alpn_string	

13.1.2.2 struct qapi_Ssl_Cert

SSL certification.

Data fields

Type	Parameter	Description
uint8_t *	pRootCa	
uint32_t	rootCaSize	
uint8_t *	pClientCert	
uint32_t	clientCertSize	
uint8_t *	pPrivateKey	
uint32_t	privateKeySize	

13.1.2.3 struct qapi_Net_HTTPc_Response_t

HTTP client response. For use with qapi_HTTPc_CB_t.

Data fields

Type	Parameter	Description
uint32_t	length	Length of the data.
uint32_t	contentlength	Length of the content.
uint32_t	resp_Code	Response code.
const uint8_t *	data	Data associated with the response if not NULL.

13.1.3 Typedef Documentation

13.1.3.1 typedef struct qapi_Ssl_Config qapi_Ssl_Config_t

SSL configuration.

13.1.3.2 typedef struct qapi_Ssl_Cert qapi_Ssl_Cert_t

SSL certification.

13.1.3.3 typedef void(* qapi_HTTPc_CB_t)(void *arg,int32_t state,void *value)

User registered callback for returning response message.

13.1.4 Enumeration Type Documentation

13.1.4.1 enum qapi_Net_HTTPc_Method_e

Supported http client methods. For use with qapi_Net_HTTPc_Request().

13.1.4.2 enum qapi_Net_HTTPc_CB_State_e

HTTP client callback state. For use with qapi_HTTPc_CB_t.

Enumerator:

QAPI_NET_HTTPC_RX_ERROR_SERVER_CLOSED Server closes the connection.

QAPI_NET_HTTPC_RX_ERROR_RX_PROCESS Size section of a chunk is longer than the Rx buffer length.

QAPI_NET_HTTPC_RX_ERROR_RX_HTTP_HEADER Header section is longer than the Rx buffer length.

QAPI_NET_HTTPC_RX_ERROR_INVALID_RESPONSECODE Status code is less than 100 or greater than 999.

QAPI_NET_HTTPC_RX_ERROR_CLIENT_TIMEOUT Request times out.

QAPI_NET_HTTPC_RX_ERROR_NO_BUFFER Reserved.

QAPI_NET_HTTPC_RX_CONNECTION_CLOSED Reserved.

QAPI_NET_HTTPC_RX_ERROR_CONNECTION_CLOSED Connection is closed due to error on socket read.

QAPI_NET_HTTPC_RX_FINISHED Response is completely received.

QAPI_NET_HTTPC_RX_MORE_DATA Response is partially received.

QAPI_NET_HTTPC_RX_TUNNEL_ESTABLISHED Tunnel to the origin server is established.

QAPI_NET_HTTPC_RX_DATA_FROM_TUNNEL Receiving data from origin server.

QAPI_NET_HTTPC_RX_TUNNEL_CLOSED Server closes the tunnel.

QAPI_NET_HTTPC_RX_CHUNK_CONTINUE Receiving CONTINUE from server

13.1.5 Function Documentation

13.1.5.1 qapi_Status_t qapi_Net_HTTPc_Start (void)

(Re)starts the HTTP client module.

Normally, this is called to start or restart the client after it was stopped via a call to qapi_Net_HTTPc_Stop().

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.2 qapi_Status_t qapi_Net_HTTPc_Stop (void)

Stops the HTTP client module.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.3 qapi_Net_HTTPc_handle_t qapi_Net_HTTPc_New_sess2 (uint32_t *timeout*, uint32_t *isHttps*, qapi_HTTPc_CB_t *callback*, void * *arg*, uint16_t *httpc_Max_Body_Length*, uint16_t *httpc_Max_Header_Length*, uint16_t *httpc_Rx_Buffer_Size*, uint16_t *ip_prefer*, uint16_t *ssl_pre_buffer*)

Starts a new a HTTP client session.

Parameters

in	*timeout*	Timeout (in ms) on an HTTP request in this session.
in	*isHttps*	Indicates whether reqeust is http or https.
in	*callback*	Pointer to the user callback function (see qapi_HTTPc_CB_t)
in	*arg*	Argument for the callback function.
in	*httpc_Max_Body_Length*	Size in bytes of message-body buffer for HTTP request.
in	*httpc_Max_Header_Length*	Size in bytes of header buffer for HTTP request.
in	*httpc_Rx_Buffer_Size*	Size in bytes of Rx buffer for HTTP response. If size is less than 512, system will use 512.
in	*ip_prefer*	Prefer to select an IP type: ipv4 or ipv6.
in	*ssl_Cfg*	ssl_pre_buffer parameters.

Returns

On success, a non-NULL handle is returned; on error, NULL is returned.

13.1.5.4 qapi_Net_HTTPc_handle_t qapi_Net_HTTPc_New_sess (uint32_t *timeout,* uint32_t *isHttps,* qapi_HTTPc_CB_t *callback,* void * *arg,* uint16_t *httpc_Max_Body_Length,* uint16_t *httpc_Max_Header_Length*)

Starts a new a HTTP client session.

Parameters

in	*timeout*	Timeout (in ms) on an HTTP request in this session.
in	*isHttps*	Indicate whether request is http or https.
in	*callback*	Pointer to the user callback function (see qapi_HTTPc_CB_t).
in	*arg*	Argument for the callback function.
in	*httpc_Max_Body_Length*	Size in bytes of message-body buffer for HTTP request.
in	*httpc_Max_Header_Length*	Size in bytes of header buffer for HTTP request.

Note

Internally, the system allocates 1,750-byte RX buffer for caller.

Returns

On success, a non-NULL handle is returned; on error, NULL is returned.

13.1.5.5 qapi_Status_t qapi_Net_HTTPc_Free_sess (qapi_Net_HTTPc_handle_t *handle*)

Frees an HTTP client session.

Disconnects from the server and frees the memory.

Parameters

in	*handle*	HTTP client session handle.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.6 qapi_Status_t qapi_Net_HTTPc_Connect (qapi_Net_HTTPc_handle_t *handle,* const char * *server,* uint16_t *port*)

Connects to an HTTP server in Blocking mode.

Parameters

in	*andle*	HTTP client session handle.
in	*server*	Pointer to server or proxy. For example, "192.168.2.100" or "www.example.com".
in	*port*	Port of the server.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.7 qapi_Status_t qapi_Net_HTTPc_Disconnect (qapi_Net_HTTPc_handle_t *handle*)

Disconnects an HTTP client session from the server.

Parameters

in	*handle*	HTTP client session handle.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.8 qapi_Status_t qapi_Net_HTTPc_Request (qapi_Net_HTTPc_handle_t *handle,* qapi_Net_HTTPc_Method_e *cmd,* const char * *URL*)

Send an HTTP request to an HTTP server or proxy.

Parameters

in	*handle*	HTTP client session handle.
in	*cmd*	HTTP request (see qapi_Net_HTTPc_Method_e)
in	*URL*	Pointer to the request URL. For example, "index.html" or "/cgi/mycgi.pl" if cmd is not QAPI_NET_HTTP_CLIENT_CONNECT_E, "www.example.com:22" if cmd is QAPI_NET_HTTP_CLIENT_CONNECT_E.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.9 qapi_Status_t qapi_Net_HTTPc_Tunnel_To_HTTPS (qapi_Net_HTTPc_handle_t *handle,* const char * *calist,* const char * *URL*)

Send an HTTP CONNECT request to a proxy for establishing a connection to an HTTPS origin server.

Parameters

in	*handle*	HTTP client session handle.
in	*calist*	A file (on the local file system) containing CA certificates, which are in SharkSSL format. This is used for authenticating the origin HTTPS server. It can be set to NULL.
in	*URL*	Pointer to the request URL. For example, "www.example.com:22".

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.10 qapi_Status_t qapi_Net_HTTPc_Set_Body (qapi_Net_HTTPc_handle_t *handle*, const char * *body*, uint32_t *body_Length*)

Sets the body on an HTTP client session.

Parameters

in	*handle*	HTTP client session handle.
in	*body*	Pointer to the body.
in	*body_Length*	Length of the body.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.11 qapi_Status_t qapi_Net_HTTPc_Set_Param (qapi_Net_HTTPc_handle_t *handle*, const char * *key*, const char * *value*)

Forms a URL-encoded string on an HTTP client session.

Parameters

in	*handle*	HTTP client session handle.
in	*key*	Pointer to the key.
in	*value*	Pointer to the value.

```
// The following calls generate a URL-encoded string which will be
// the message body for POST request or
// the query string for GET request.
qapi_Net_HTTPc_Set_Param(handle, "name", "Lucy");
qapi_Net_HTTPc_Set_Param(handle, "neighbors", "Fred & Ethel");

// For example, if user calls
// qapi_Net_HTTPc_Request(handle, QAPI_NET_HTTP_CLIENT_GET_E,
"index.html");
// the start line, "GET /index.html?name=Lucy&neighbors=Fred+%26+Ethel
HTTP/1.1\r\n",
// is generated.
```

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.12 qapi_Status_t qapi_Net_HTTPc_Add_Header_Field (qapi_Net_HTTPc_handle_t *handle*, const char * *type*, const char * *value*)

Sets the header field for an HTTP client session.

Parameters

in	*handle*	HTTP client session handle.
in	*type*	Pointer to the type.
in	*value*	Pointer to the value.

Note

If this API is not called, the system will send the following headers for user:

- Host: <hostname>:<port>
- Accept: text/html, <asterisk>/<asterisk>
- User-Agent: IOE Client
- Connection: keep-alive
- Cache-control: no-cache

  Additionally, for POST, PUT, and PATCH requests if message body has data:

- Content-length: <nn>
- Content-Type: application/x-www-form-urlencoded

  (for POST request)

If this API is called, the system will send the following headers for the user:

- Host: <hostname>:<port>
- Connection: keep-alive
- User's own headers added by calling this API Additionally, for POST, PUT, and PATCH requests if message body has data:
- Content-length: <nn>

```
 // The following calls will generate request headers in an HTTP GET
request:
 // "User-Agent: My Own Browser 1.0\r\n"
 // "Connection: keep-alive\r\n"
 qapi_Net_HTTPc_Add_Header_Field(handle, "User-Agent", "My Own Browser
1.0");
 qapi_Net_HTTPc_Add_Header_Field(handle, "Connection", "keep-alive");
 qapi_Net_HTTPc_Request(handle, QAPI_NET_HTTP_CLIENT_GET_E, "index.html"
);
```

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.13 qapi_Status_t qapi_Net_HTTPc_Clear_Header (qapi_Net_HTTPc_handle_t *handle*)

Clears the header field for an HTTP client session.

Parameters

in	*handle*	HTTP client session handle.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.14 qapi_Status_t qapi_Net_HTTPc_Configure_SSL (qapi_Net_HTTPc_handle_t *handle*, qapi_Ssl_Config_t * *ssl_Cfg*)

Sets SSL configuration parameters on a secure (HTTPS) session.

Parameters

in	*handle*	HTTP client session handle.
in	*ssl_Cfg*	SSL connection parameters.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.15 qapi_Status_t qapi_Net_HTTPc_Configure_Cert (qapi_Net_HTTPc_handle_t *handle*, qapi_Ssl_Cert_t * *ssl_Cfg*)

Sets SSL certificate parameters on a secure (i.e., HTTPS) session.

Parameters

in	*handle*	HTTP client session handle.
in	*ssl_Cfg*	SSL certificate parameters.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.16 qapi_Status_t qapi_Net_HTTPc_CB_Enable_Adding_Header (qapi_Net_HTTPc_handle_t *handle*, uint16_t *enable*)

Enables/disables the addition of an HTTP head in a session callback.

By default, the system returns the message body of the response via user registered callback (see qapi_HTTPc_CB_t). Message headers are not returned. To enable the system to also return messages headers, this API should be called with 'enable'= 1.

Parameters

in	*handle*	HTTP client session handle.
in	*enable*	1 – Enabled. 0 – Disabled.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.17 qapi_Status_t qapi_Net_HTTPc_Send_Data (qapi_Net_HTTPc_handle_t *handle*, const char * *buf*, uint32_t *length*)

Sends raw data when an HTTP tunnel is established.

Parameters

in	*handle*	HTTP client session handle.
in	*buf*	Pointer to data.
in	*length*	Length of data.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

13.1.5.18 qapi_Status_t qapi_Net_HTTPc_Send_Chunk (qapi_Net_HTTPc_handle_t *handle,* qapi_Net_HTTPc_Method_e *cmd,* const char * *URL,* const char * *chunk,* uint32_t *chunk_size,* uint8_t *chunk_flag,* int32_t *total_size*)

Sends chunk data.

Parameters

in	*handle*	HTTP client session handle.
in	*cmd*	HTTP request (see qapi_Net_HTTPc_Method_e)
in	*URL*	Pointer to the request URL. For example, "index.html" or "/cgi/mycgi.pl".
in	*chunk*	Pointer to chunk data.
in	*chunk_size*	Length of chunk data.
in	*chunk_type*	0x00 – Non chunk encoded without http header. 0x01 – Non chunk encoded with http header (first pakcet). 0x80 – Chunk encoded without http header. 0x81 – Chunk encoded with http header.

Returns

When all data is sent, 0 is returned. On error, non-zero is returned.

13.1.5.19 qapi_Status_t qapi_Net_HTTPc_Rlease_Pre_allocate_buffer (void)

Release pre-allocated buffer.

Returns

On success, 0 is returned. On error, QAPI_NET_STATUS_HTTPC_xxx is returned.

14 HTTP Server

14.1 HTTP server

14.1.1 Function Documentation

14.1.1.1 qapi_Status_t qapi_get_wifi_cfg (char * *ssid,* char * *password*)

Gets the Wi-Fi configuration.

Parameters

ssid	SSID.*
password	Password.

14.1.1.2 qapi_Status_t qapi_web_start (uint16_t *server_port*)

Starts the web server.

Parameters

server_port	Server port.

14.1.1.3 qapi_Status_t qapi_web_stop (void)

Stops the web server.

15 UART

15.1 UART

15.1.1 Define Documentation

15.1.1.1 #define QAPI_I2CM_ERROR __QAPI_ERROR(QAPI_MOD_I2C, 1)

Common error.

15.1.1.2 #define QAPI_UART_ERROR __QAPI_ERROR(QAPI_MOD_UART, 1)

Common error.

15.1.1.3 #define QAPI_UART_ERROR_NULL_PTR_ERROR __QAPI_ERROR(QAPI_MOD_UART, 2)

NULL pointer error.

15.1.1.4 #define QAPI_UART_ERROR_INVALID_PARAM __QAPI_ERROR(QAPI_MOD_UART, 3)

Invalid parameter error.

15.1.1.5 #define QAPI_UART_ERROR_CFG_PARAM __QAPI_ERROR(QAPI_MOD_UART, 4)

Configuration error.

15.1.1.6 #define QAPI_UART_ERROR_BAUDRATE_CFG __QAPI_ERROR(QAPI_MOD_UART, 5)

Baudrate configuration error.

15.1.1.7 #define QAPI_UART_ERROR_SEND_BUSY __QAPI_ERROR(QAPI_MOD_UART, 6)

Send busy error.

15.1.1.8 #define QAPI_UART_ERROR_RECV_BUSY __QAPI_ERROR(QAPI_MOD_UART, 7)

Receive busy error.

15.1.1.9 #define QAPI_UART_ERROR_TX_ENQUEUE_SEM_SYNC __QAPI_ERROR(QAPI_MOD_UART, 8)

Tx enqueue sem sync error.

15.1.1.10 #define QAPI_UART_ERROR_TX_ENQUEUE_FULL __QAPI_ERROR(QAPI_MOD_UART, 9)

Tx enqueue full error.

15.1.1.11 #define QAPI_UART_ERROR_TX_DEQUEUE_EMPTY __QAPI_ERROR(QAPI_MOD_UART, 10)

Tx dequeue empty error.

15.1.1.12 #define QAPI_UART_ERROR_RX_ENQUEUE_FULL __QAPI_ERROR(QAPI_MOD_UART, 11)

Rx enqueue full error.

15.1.1.13 #define QAPI_UART_ERROR_RX_DEQUEUE_SEM_SYNC __QAPI_ERROR(QAPI_MOD_UART, 12)

Rx dequeue sem error.

15.1.1.14 #define QAPI_UART_ERROR_RX_DEQUEUE_EMPTY __QAPI_ERROR(QAPI_MOD_UART, 13)

Rx dequeue empty error.

15.1.1.15 #define QAPI_UART_ERROR_TRANSFER_TIMEOUT __QAPI_ERROR(QAPI_MOD_UART, 14)

Transfer timeout error.

15.1.1.16 #define QAPI_UART_ERROR_INPUT_FIFO_UNDER_RUN __QAPI_ERROR(QAPI_MOD_UART, 15)

Input fifo under run error.

15.1.1.17 #define QAPI_UART_ERROR_INPUT_FIFO_OVER_RUN __QAPI_ERROR(QAPI_MOD_UART, 16)

Input fifo over run error.

15.1.1.18 #define QAPI_UART_ERROR_OUTPUT_FIFO_UNDER_RUN __QAPI_ERROR(QAPI_MOD_UART, 17)

Output fifo under run error.

15.1.1.19 #define QAPI_UART_ERROR_OUTPUT_FIFO_OVER_RUN __QAPI_ERROR(QAPI_MOD_UART, 18)

Output fifo over run error.

15.1.1.20 #define QAPI_UART_ERROR_TRANSFER_FORCE_TERMINATED __QAPI_ERROR(QAPI_MOD_UART, 19)

Transfer force terminated.

15.1.1.21 #define QAPI_UART_ERROR_BUS_CLK_CFG_FAIL __QAPI_ERROR(QAPI_MOD_UART, 20)

Bus clock configuration failure.

15.1.1.22 #define QAPI_UART_ERROR_BUS_GPIO_ENABLE_FAIL __QAPI_ERROR(-QAPI_MOD_UART, 21)

Bus GPIO enable failire.

15.1.1.23 #define QAPI_UART_ERROR_CANCEL_TRANSFER_FAIL __QAPI_ERROR(-QAPI_MOD_UART, 22)

Cancel transfer failure.

15.1.1.24 #define QAPI_UART_ERROR_BOOTSTRAP_CFG_FAIL __QAPI_ERROR(QAPI_MOD_UART, 23)

Bootstrap configuration failure.

15.1.1.25 #define QAPI_UART_ERROR_DEVICE_STATE __QAPI_ERROR(QAPI_MOD_UART, 23)

Device state is invalid for the operation.

15.1.2 Data Structure Documentation

15.1.2.1 struct qapi_UART_Open_Config_t

Structure for UART configuration.

Data fields

Type	Parameter	Description
uint32_t	baud_Rate	Baud rates
qapi_UART_Parity_Mode_e	parity_Mode	Parity mode.
qapi_UART_Bits_Per_Char_e	bits_Per_Char	Bits per character.
qapi_UART_Num_Stop_Bits_e	num_Stop_Bits	Number of stop bits.
qbool_t	enable_Loopback	Enable loopback.

15.1.3 Enumeration Type Documentation

15.1.3.1 enum qapi_UART_Instance_t

UART port ID enumeration.

Enumeration to specify which port is to be opened during the uart_open call.

Enumerator:

QAPI_UART_INST_0 port0.

15.1.3.2 enum qapi_UART_Bits_Per_Char_e

Enumeration to specify how many UART bits are to be used per character configuration.

Enumerator:

QAPI_UART_5_BITS_PER_CHAR_E 5 bits per character. Due to hardware limitation, currently not supported.
QAPI_UART_6_BITS_PER_CHAR_E 6 bits per character. Due to hardware limitation, currently not supported.
QAPI_UART_7_BITS_PER_CHAR_E 7 bits per character. Due to hardware limitation, currently not supported.
QAPI_UART_8_BITS_PER_CHAR_E 8 bits per character.

15.1.3.3 enum qapi_UART_Num_Stop_Bits_e

Enumeration for the UART number of stop bits configuration.

Enumerator:

QAPI_UART_1_0_STOP_BITS_E 1.0 stop bit.
QAPI_UART_1_5_OR_2_0_STOP_BITS_E 1.5 stop bits for 5 bits data, otherwise 2.0 stop bits.

15.1.3.4 enum qapi_UART_Parity_Mode_e

Enumeration for the UART parity mode configuration.

Enumerator:

QAPI_UART_NO_PARITY_E No parity.
QAPI_UART_ODD_PARITY_E Odd parity.
QAPI_UART_EVEN_PARITY_E Even parity.

15.1.4 Function Documentation

15.1.4.1 qapi_Status_t qapi_UART_Close (qapi_UART_Instance_t *instance*)

Closes the UART port. Not to be called from ISR context.

Releases clock, interrupt, and GPIO handles related to this UART and cancels any pending transfers.

NOTE Do not call this API from ISR context.

Parameters

in	*handle*	UART handle provided by qapi_UART_Open().

Returns

QAPI_OK Port close successful.
QAPI_ERR_XX Port close failed. For error code, refer to api_Status_t.

15.1.4.2 qapi_Status_t qapi_UART_Open (qapi_UART_Instance_t *instance,* qapi_UART_Open_Config_t * *config*)

Initializes the UART port. Not to be called from ISR context.

Opens the UART port and configures the corresponding clocks, interrupts, and GPIO.

NOTE Do not call this API from ISR context.

Parameters

in	*id*	ID of the port to be opened.
in	*config*	Structure that holds all configuration data.

Returns

QAPI_OK Port open successful.
QAPI_ERR_XX Port open failed. For error code, refer to api_Status_t.

15.1.4.3 qapi_Status_t qapi_UART_Receive (qapi_UART_Instance_t *instance,* char * *buf,* uint32_t *buf_Size,* uint32_t * *recved*)

Queues the buffer provided for receiving the data. Not to be called from ISR context.

This is an synchronous call. Return when the transaction is done.

Call uart_receive immediately after uart_open to queue a buffer.

NOTE Do not call this API from ISR context.

Parameters

in	*handle*	UART handle provided by qapi_UART_Open().
in	*buf*	Buffer to be filled with data.
in	*buf_Size*	Buffer size. Must be ≥ 4 and a multiple of 4.
in	*recved*	Callback data to be passed when rx_cb_isr is called during Rx completion.

Returns

QAPI_OK Recieve transcation was successful.
QAPI_ERR_XX Receive transcation has error. For error code, refer to api_Status_t.

15.1.4.4 qapi_Status_t qapi_UART_Transmit (qapi_UART_Instance_t *instance,* char * *buf,* uint32_t *bytes_To_Tx,* uint32_t * *sent*)

Transmits data from a specified buffer. Not to be called from ISR context.

This is an synchronous call. Return when transmit is completed.

The buffer is owned by the UART driver until the call returns.

NOTE Do not call this API from ISR context.

Parameters

in	*handle*	UART handle provided by qapi_UART_Open().
in	*buf*	Buffer with data for transmit.
in	*bytes_To_Tx*	Bytes of data to transmit.
in	*sent*	Bytes of data sent.

Returns

QAPI_OK Transmit was successful.
QAPI_ERROR Transmit buffer failed. For error code, refer to api_Status_t.

15.1.4.5 qapi_Status_t qapi_UART_Open_With_Rx_Timeout (qapi_UART_Instance_t *instance,* qapi_UART_Open_Config_t * *config,* uint32_t *timeout*)

Initializes the UART port.

Opens the UART port and configures the corresponding clocks, interrupts, and GPIO with Rx timeout.

NOTE Do not call this API from ISR context.

Parameters

in	*instance*	ID of the port to be opened.
in	*config*	Structure that holds all configuration data.

in	*timeout*	Timeout of Rx thread.

Returns

- QAPI_OK: Port open successful.
- QAPI_ERR_XX: Port open failed, for error codes see #api_Status_t.

15.1.5 Variable Documentation

15.1.5.1 uint32_t qapi_UART_Open_Config_t::baud_Rate

Baud rates

15.1.5.2 qapi_UART_Parity_Mode_e qapi_UART_Open_Config_t::parity_Mode

Parity mode.

15.1.5.3 qapi_UART_Bits_Per_Char_e qapi_UART_Open_Config_t::bits_Per_Char

Bits per character.

15.1.5.4 qapi_UART_Num_Stop_Bits_e qapi_UART_Open_Config_t::num_Stop_Bits

Number of stop bits.

15.1.5.5 qbool_t qapi_UART_Open_Config_t::enable_Loopback

Enable loopback.

16 PRNG

16.1 PRNG APIs

16.1.1 Define Documentation

16.1.1.1 #define qapi_prng_get nt_wlan_hw_prng_get

16.1.2 Function Documentation

16.1.2.1 uint8_t qapi_prng_get (uint8_t * *ptr,* uint16_t *len*)

Gets the PRNG status.

Parameters

in	*ptr*	
in	*len*	

17 RRAM

17.1 RRAM APIs

17.1.1 Define Documentation

17.1.1.1 #define RRAM_DEVICE_DONE 0

Operation passed.

17.1.1.2 #define RRAM_DEVICE_FAIL (-1)

Operation failed.

17.1.2 Data Structure Documentation

17.1.2.1 struct IDAddr

Definition structure with RRAM ID and address

Data fields

Type	Parameter	Description
uint32_t	id	RRAM ID.
uint32_t	addr	RRAM address.

17.1.3 Enumeration Type Documentation

17.1.3.1 enum rram_status_t

Enumerator:

RRAM_OFFSET_ERROR
RRAM_ADDRESS_ERROR
RRAM_OK

17.1.4 Function Documentation

17.1.4.1 qapi_Status_t qapi_rram_read (uint32_t *partid,* uint32_t *offset,* uint8_t * *buffer,* uint32_t *len*)

Reads data from RRAM.

Parameters

in	*partid*	The part ID to map the UD part base address in the secure boot loader.
in	*offset*	The RRAM address to start to read from.

in	*len*	Number of bytes to read.
out	*buffer*	Data buffer for a RRAM read operation.

Returns

- QAPI_OK: Read completed successfully.
- Error code: An error occurred.

17.1.4.2 qapi_Status_t qapi_rram_write (uint32_t *partid,* uint32_t *offset,* uint8_t * *buffer,* uint32_t *len*)

Writes data to RRAM.

Parameters

in	*partid*	The part ID to map the UD part base address in the secure boot loader.
in	*offset*	The RRAM address to start to write to.
in	*len*	Number of bytes to write.
in	*buffer*	Data buffer containing data to be written.

Returns

- QAPI_OK: Write completed successfully.
- Error code: An error occurred.

18 RTC

18.1 RTC APIs

QAPIs to manually set and get the time in Julian format or NTP format.

18.1.1 Define Documentation

18.1.1.1 #define NUM_DAYS_PER_YEAR 365

18.1.1.2 #define DIFF_SEC_1900_1970 (2208988800UL)

18.1.2 Data Structure Documentation

18.1.2.1 struct qapi_Time_s

Time in Julian format.

Data fields

Type	Parameter	Description
uint16_t	year	Year [1980 through 2100].
uint16_t	month	Month of the year [1 through 12].
uint16_t	day	Day of the month [1 through 31].
uint16_t	hour	Hour of the day [0 through 23].
uint16_t	minute	Minute of the hour [0 through 59].
uint16_t	second	Second of the minute [0 through 59].
uint16_t	day_Of_Week	Day of the week [0 through 6] or [Monday through Sunday].

18.1.2.2 struct ntp_Time_s

Time in NTP format.

Data fields

Type	Parameter	Description
uint32_t	second	
uint32_t	frac	

18.1.2.3 struct time_zone_s

Time zone.

Data fields

Type	Parameter	Description
uint8_t	hour	

Type	Parameter	Description
uint8_t	min	
uint8_t	add_sub	

18.1.3 Typedef Documentation

18.1.3.1 typedef struct qapi_Time_s qapi_Time_t

Time in Julian format.

18.1.3.2 typedef struct ntp_Time_s ntp_Time_t

Time in NTP format.

18.1.3.3 typedef struct time_zone_s time_zone_t

Time zone.

18.1.4 Function Documentation

18.1.4.1 qapi_Status_t qapi_Core_RTC_Julian_Get (qapi_Time_t * *tm*)

Gets the Julian time.

Parameters

in	*tm*	Pointer to a buffer to contain the Julian time.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.2 qapi_Status_t qapi_Core_RTC_Julian_Set (qapi_Time_t * *tm*)

Sets the Julian time.

Parameters

in	*tm*	Pointer to a buffer to contain the Julian time.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.3 qapi_Status_t qapi_Core_RTC_NTP_Get (ntp_Time_t * *tm*)

Gets the NTP time.

Parameters

in	*ms*	Pointer to a buffer to contain the NTP time.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.4 qapi_Status_t qapi_Core_RTC_NTP_Set (ntp_Time_t * *tm*)

Sets the NTP time.

Parameters

in	*ms*	Pointer to a buffer to contain NTP time.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.5 qapi_Status_t qapi_Core_Time_Zone_Get (time_zone_t * *zone*)

Gets the time zone.

Parameters

in	*zone*	Pointer to a buffer to contain the time zone.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.6 qapi_Status_t qapi_Core_Time_Zone_Set (time_zone_t * *zone*)

Sets the time zone.

Parameters

in	*zone*	Pointer to a buffer to contain the time zone.

Returns

- QAPI_OK on success
- Error code on failure

18.1.4.7 qapi_Status_t qapi_Core_Obtain_Boot_Reason (uint32_t * *data*)

Gets the boot reason.

Parameters

in	*data*	Pointer to a uint32_t to contain boot reason.

Returns

- QAPI_OK on success
- Error code on failure

19 Common APIs

The following information is common to all QAPI modules.

- Build information – QAPI/SDK version and build information.
- Status information
- Error code formats – Definitions used to format error codes based on their module.

Error codes that use these macros will be a negative value of the format -((10000 ∗ <Module ID>) + <Status Code>).

- Module IDs – Definitions representing the IDs for the various QAPI modules. If adding your own module IDs, use IDs starting from 100 to avoid conflicts with future module updates and additions.
- Status codes – Definitions representing status codes common to all QAPI modules.
- qapi_cmn_types – Custom QAPI type information.

19.1 Build information

19.1.1 Define Documentation

19.1.1.1 #define QAPI_CHIP_VERSION 0x0101

Chip version.

19.1.1.2 #define QAPI_BUILD_VERSION 0x07D0

Build version.

19.1.1.3 #define QAPI_CHIP_VERSION_STR "0101"

Chip version string.

19.1.1.4 #define QAPI_BUILD_VERSION_STR "07D0"

Build version string.

19.1.1.5 #define QAPI_VERSION_MAJOR (3)

Major version.

19.1.1.6 #define QAPI_VERSION_MINOR (0)

Minor version.

19.1.1.7 #define QAPI_VERSION_NIT (0)

Version NIT.

19.1.1.8 #define __QAPI_VERSION_MAJOR_MASK (0xff000000)

Version major mask.

19.1.1.9 #define __QAPI_VERSION_MINOR_MASK (0x00ff0000)

Version minor mask.

19.1.1.10 #define __QAPI_VERSION_NIT_MASK (0x0000ffff)

Version NIT mask.

19.1.1.11 #define __QAPI_VERSION_MAJOR_SHIFT (24)

Version major shift.

19.1.1.12 #define __QAPI_VERSION_MINOR_SHIFT (16)

Version minor shift.

19.1.1.13 #define __QAPI_VERSION_NIT_SHIFT (0)

Version NIT shift.

19.1.1.14 #define __QAPI_ENCODE_VERSION(*__major__, __minor__, __nit__*)

Value:

```
(((__major__) << __QAPI_VERSION_MAJOR_SHIFT) | \
                                                ((__minor__) <<
       __QAPI_VERSION_MINOR_SHIFT) | \
                                                ((__nit__)   <<
       __QAPI_VERSION_NIT_SHIFT))
```

QAPI encode version.

19.1.2 Data Structure Documentation

19.1.2.1 struct qapi_FW_Info_t

Data structure used by application to get build information.

Data fields

Type	Parameter	Description
uint32_t	qapi_Version_-Number	QAPI version number.
uint32_t	crm_Build_-Number	CRM build number.
char	crm_full_-version[16]	CRM build full version.

19.1.3 Function Documentation

19.1.3.1 qapi_Status_t qapi_Get_FW_Info (qapi_FW_Info_t * *info*)

Retrieves version information from the system.

Parameters

out	*info*	Value retrieved from system.

Returns

QAPI_OK – Requested parameter retrieved from the system.
Non-Zero value – Parameter retrieval failed.

Dependencies

None.

19.2 Status information

19.2.1 Typedef Documentation

19.2.1.1 typedef int32_t qapi_Status_t

Status of an operation.

19.3 Error code formats

The following definitions are used to format error codes based on their module. Error codes that use these macros will be a negative value of the format -((10000 * <Module ID>) + <Status Code>).

19.3.1 Define Documentation

19.3.1.1 #define __QAPI_ERR_MOD_OFFSET (10000)

Module offset error code format.

19.3.1.2 #define __QAPI_ERR_ENCAP_MOD_ID(*__mod_id__*) ((__mod_id__) * __QAPI_ERR_MOD_OFFSET)

Module offset and module ID error code format.

19.3.1.3 #define __QAPI_ERROR(*__mod_id__, __err__*) ((qapi_Status_t)(0 - (__QAPI_ERR_ENCAP_MOD_ID(__mod_id__) + (__err__))))

Module offset, module ID, and status code error code format.

19.4 Module IDs

The following definitions represent the IDs for the various modules of the QAPI.

If you are an OEM that wants to add your own module IDs, Qualcomm® recommend starting IDs from 100. This will avoid possible conflicts with future module updates and additions to the QAPI.

19.4.1 Define Documentation

19.4.1.1 #define QAPI_MOD_BASE (1)

Base module ID.

19.4.1.2 #define QAPI_MOD_UART (2)

UART module ID.

19.4.1.3 #define QAPI_MOD_I2C (3)

I2C module ID.

19.4.1.4 #define QAPI_MOD_SPI (4)

SPI module ID.

19.4.1.5 #define QAPI_MOD_GPIO (5)

GPIO module ID.

19.4.1.6 #define QAPI_MOD_FTC (6)

FTC module ID.

19.4.1.7 #define QAPI_MOD_M2MDMA (7)

M2MDMA module ID.

19.4.1.8 #define QAPI_MOD_TMR (8)

TMR module ID.

19.4.1.9 #define QAPI_MOD_CRYPTO (9)

CRYPTO module ID.

19.4.1.10 #define QAPI_MOD_LIC (10)

LIC module ID.

19.4.1.11 #define QAPI_MOD_FWUP (11)

FWUP module ID.

19.4.1.12 #define QAPI_MOD_NVM (12)

NVM module ID.

19.4.1.13 #define QAPI_MOD_APPI2C (13)

APPI2C module ID.

19.4.1.14 #define QAPI_MOD_CONSOLE (14)

Console module ID.

19.4.1.15 #define QAPI_MOD_WIFI (15)

Wi-Fi module ID.

19.4.1.16 #define QAPI_MOD_NETWORKING (16)

NET module ID.

19.4.1.17 #define QAPI_MOD_FLASH (17)

Flash module ID.

19.4.1.18 #define QAPI_MOD_HKADC (18)

HKADC module ID.

19.5 Status codes

The following definitions represent status codes common to all QAPI modules.

19.5.1 Define Documentation

19.5.1.1 #define QAPI_OK ((qapi_Status_t)(0))

Success.

19.5.1.2 #define QAPI_ERROR (__QAPI_ERROR(QAPI_MOD_BASE, 1))

General error.

19.5.1.3 #define QAPI_ERR_INVALID_PARAM (__QAPI_ERROR(QAPI_MOD_BASE, 2))

Invalid parameter.

19.5.1.4 #define QAPI_ERR_NO_MEMORY (__QAPI_ERROR(QAPI_MOD_BASE, 3))

Memory allocation error.

19.5.1.5 #define QAPI_ERR_NO_RESOURCE (__QAPI_ERROR(QAPI_MOD_BASE, 4))

Resource allocation error.

19.5.1.6 #define QAPI_ERR_BUSY (__QAPI_ERROR(QAPI_MOD_BASE, 6))

Operation is busy.

19.5.1.7 #define QAPI_ERR_NO_ENTRY (__QAPI_ERROR(QAPI_MOD_BASE, 7))

Entry was not found.

19.5.1.8 #define QAPI_ERR_NOT_SUPPORTED (__QAPI_ERROR(QAPI_MOD_BASE, 8))

Feature is not supported.

19.5.1.9 #define QAPI_ERR_TIMEOUT (__QAPI_ERROR(QAPI_MOD_BASE, 9))

Operation timed out.

19.5.1.10 #define QAPI_ERR_BOUNDS (__QAPI_ERROR(QAPI_MOD_BASE, 10))

Out of bounds.

19.5.1.11 #define QAPI_ERR_BAD_PAYLOAD (__QAPI_ERROR(QAPI_MOD_BASE, 11))

Bad payload.

19.5.1.12 #define QAPI_ERR_EXISTS (__QAPI_ERROR(QAPI_MOD_BASE, 12))

Entry already exists.

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics, and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates, and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates, and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates, and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES, AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools, and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the Website Terms of Use on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation, any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws and principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other products and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc., and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.

THE DOCUMENTATION ACCOMPANYING THE MATERIALS AND/OR RELEVANT PRODUCTS AND SERVICE OFFERINGS MAY INCLUDE IMPORTANT USE LIMITATIONS. ANY DEVIATIONS FROM APPLICABLE USE LIMITATIONS MAY ADVERSELY IMPACT PERFORMANCE, DURABILITY, QUALITY OR SAFETY. YOU ASSUME ALL RISKS AND LIABILITIES ASSOCIATED WITH ANY DEVIATIONS