Qualcomm

Qualcomm Technologies, Inc.

QCC730 API Specification

API Reference

80-Y9730-4 Rev. AG

May 21, 2025

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.

Contents

1 Introduction **23**
1.1 Purpose . . . 23
1.2 Conventions . . . 23
1.3 Technical Assistance . . . 23

2 Functional overview **24**
3.1 Console APIs . . . 25

3 Console module **25**
3.1.1 Data Structure Documentation . . . 25
3.1.1.1 struct QAPI_Console_Parameter_s . . . 25
3.1.1.2 struct QAPI_Console_Command_s . . . 25
3.1.1.3 struct QAPI_Console_Command_Group_s . . . 25
3.1.2 Typedef Documentation . . . 26
3.1.2.1 QAPI_Console_Handle_t . . . 26
3.1.2.2 QAPI_Console_Group_Handle_t . . . 26
3.1.2.3 QAPI_Console_Parameter_t . . . 26
3.1.2.4 QAPI_Concole_Command_Function_t . . . 26
3.1.2.5 QAPI_Console_Command_t . . . 26
3.1.2.6 QAPI_Console_Command_Group_t . . . 26
3.1.3 Function Documentation . . . 26
3.1.3.1 QAPI_Console_Register_Command_Group . . . 26
3.1.3.2 QAPI_Console_Register_Command . . . 27
3.2 Console firmware upgrade codes . . . 28
3.2.1 Define Documentation . . . 28
3.2.1.1 QAPI_ERROR_CONSOLE_COMMAND_STATUS_ERROR . . . 28
3.2.1.2 QAPI_ERROR_CONSOLE_COMMAND_STATUS_USAGE . . . 28
3.2.1.3 QAPI_ERROR_CONSOLE_INIT_FAIL . . . 28
3.2.1.4 QAPI_ERROR_CONSOLE_REGISTER_CMD_GROUP_FAIL . . . 28
3.2.1.5 QAPI_ERROR_CONSOLE_REGISTER_CMD_FAIL . . . 28
4.1 Firmware upgrade APIs . . . 29

4 Firmware Upgrade module **29**
4.1.1 Data Structure Documentation . . . 29
4.1.1.1 struct qapi_Fw_Upgrade_Plugin_t . . . 29
4.1.2 Typedef Documentation . . . 29
4.1.2.1 qapi_Part_Hdl_t . . . 29
4.1.2.2 qapi_Fw_Upgrade_CB_t . . . 30
4.1.2.3 qapi_Fw_Upgrade_Plugin_Init_t . . . 30
4.1.2.4 qapi_Fw_Upgrade_Plugin_Fin_t . . . 30

4.1.2.5 qapi_Fw_Upgrade_Plugin_Recv_Data_t . . . 30
4.1.2.6 qapi_Fw_Upgrade_Plugin_Abort_t . . . 31
4.1.2.7 qapi_Fw_Upgrade_Plugin_Resume_t . . . 31
4.1.3 Enumeration Type Documentation . . . 31
4.1.3.1 qapi_Fw_Upgrade_State . . . 31
4.1.4 Function Documentation . . . 32
4.1.4.1 qapi_Fw_Upgrade . . . 32
4.1.4.2 qapi_Fw_Upgrade_Cancel . . . 33
4.1.4.3 qapi_Fw_Upgrade_Suspend . . . 33
4.1.4.4 qapi_Fw_Upgrade_Resume . . . 33
4.1.4.5 qapi_Fw_Upgrade_Done . . . 33
4.1.4.6 qapi_Fw_Upgrade_init . . . 34
4.1.4.7 qapi_Fw_Upgrade_Erase_FWD . . . 34
4.1.4.8 qapi_Fw_Upgrade_Get_FWD_Magic . . . 34
4.1.4.9 qapi_Fw_Upgrade_Set_FWD_Magic . . . 34
4.1.4.10 qapi_Fw_Upgrade_Get_FWD_Rank . . . 35
4.1.4.11 qapi_Fw_Upgrade_Get_FWD_Version . . . 35
4.1.4.12 qapi_Fw_Upgrade_Get_FWD_Status . . . 35
4.1.4.13 qapi_Fw_Upgrade_Get_FWD_Total_Images . . . 36
4.1.4.14 qapi_Fw_Upgrade_Get_Active_FWD . . . 36
4.1.4.15 qapi_Fw_Upgrade_Close_Partition . . . 36
4.1.4.16 qapi_Fw_Upgrade_First_Partition . . . 37
4.1.4.17 qapi_Fw_Upgrade_Next_Partition . . . 37
4.1.4.18 qapi_Fw_Upgrade_Find_Partition . . . 37
4.1.4.19 qapi_Fw_Upgrade_Get_Image_ID . . . 38
4.1.4.20 qapi_Fw_Upgrade_Get_Image_Version . . . 38
4.1.4.21 qapi_Fw_Upgrade_Get_Partition_Size . . . 38
4.1.4.22 qapi_Fw_Upgrade_Set_Image_Size . . . 38
4.1.4.23 qapi_Fw_Upgrade_Get_Partition_Start . . . 39
4.1.4.24 qapi_Fw_Upgrade_Get_Partition_FWD . . . 39
4.1.4.25 qapi_Fw_Upgrade_Erase_Partition . . . 40
4.1.4.26 qapi_Fw_Upgrade_Write_Partition . . . 40
4.1.4.27 qapi_Fw_Upgrade_Read_Partition . . . 40
4.2 FW upgrade definitions . . . 42
4.2.1 Define Documentation . . . 42
4.2.1.1 QAPI_FW_UPGRADE_FLAG_AUTO_REBOOT . . . 42
4.2.1.2 QAPI_FW_UPGRADE_FLAG_DUPLICATE_ACTIVE_FS . . . 42
4.2.1.3 QAPI_FW_UPGRADE_FWD_BIT_GOLDEN . . . 42
4.2.1.4 QAPI_FW_UPGRADE_FWD_BIT_CURRENT . . . 42
4.2.1.5 QAPI_FW_UPGRADE_FWD_BIT_TRIAL . . . 42
4.2.2 Variable Documentation . . . 42
4.2.2.1 fw_Upgrade_Plugin_Init . . . 42
4.2.2.2 fw_Upgrade_Plugin_Recv_Data . . . 42
4.2.2.3 fw_Upgrade_Plugin_Abort . . . 42
4.2.2.4 fw_Upgrade_Plugin_Resume . . . 42
4.2.2.5 fw_Upgrade_Plugin_Fin . . . 43
4.3 FWD bit information . . . 44
4.4 FWD boot type information . . . 45
4.4.1 Define Documentation . . . 45
4.4.1.1 QAPI_FW_UPGRADE_FWD_BOOT_TYPE_GOLDEN . . . 45

4.4.1.2 QAPI_FW_UPGRADE_FWD_BOOT_TYPE_CURRENT 45
4.4.1.3 QAPI_FW_UPGRADE_FWD_BOOT_TYPE_TRIAL 45
4.5 FWUP return status information 46
4.5.1 Define Documentation 46
4.5.1.1 QAPI_FW_UPGRADE_ERROR 46
4.5.1.2 QAPI_FW_UPGRADE_ERR_INVALID_PARAM 46
4.5.1.3 QAPI_FW_UPGRADE_ERR_NOT_INIT 46
4.5.1.4 QAPI_FW_UPGRADE_ERR_INCOMPLETE 46
4.5.1.5 QAPI_FW_UPGRADE_ERR_SESSION_IN_PROGRESS 46
4.5.1.6 QAPI_FW_UPGRADE_ERR_SESSION_NOT_START 46
4.5.1.7 QAPI_FW_UPGRADE_ERR_SESSION_NOT_READY_FOR_SUSPEND 46
4.5.1.8 QAPI_FW_UPGRADE_ERR_SESSION_NOT_SUSPEND 46
4.5.1.9 QAPI_FW_UPGRADE_ERR_SESSION_RESUME_NOT_SUPPORT . 46
4.5.1.10 QAPI_FW_UPGRADE_ERR_SESSION_CANCELLED 46
4.5.1.11 QAPI_FW_UPGRADE_ERR_SESSION_SUSPEND 46
4.5.1.12 QAPI_FW_UPGRADE_ERR_INTERFACE_NAME_TOO_LONG 47
4.5.1.13 QAPI_FW_UPGRADE_ERR_URL_TOO_LONG 47
4.5.1.14 QAPI_FW_UPGRADE_ERR_FLASH_NOT_SUPPORT_FW_UPGRADE 47
4.5.1.15 QAPI_FW_UPGRADE_ERR_FLASH_INIT_TIMEOUT 47
4.5.1.16 QAPI_FW_UPGRADE_ERR_FLASH_READ_FAIL 47
4.5.1.17 QAPI_FW_UPGRADE_ERR_FLASH_WRITE_FAIL 47
4.5.1.18 QAPI_FW_UPGRADE_ERR_FLASH_ERASE_FAIL 47
4.5.1.19 QAPI_FW_UPGRADE_ERR_FLASH_NOT_ENOUGH_SPACE 47
4.5.1.20 QAPI_FW_UPGRADE_ERR_FLASH_CREATE_PARTITION 47
4.5.1.21 QAPI_FW_UPGRADE_ERR_FLASH_IMAGE_NOT_FOUND 47
4.5.1.22 QAPI_FW_UPGRADE_ERR_FLASH_ERASE_PARTITION 47
4.5.1.23 QAPI_FW_UPGRADE_ERR_FLASH_WRITE_PARTITION 47
4.5.1.24 QAPI_FW_UPGRADE_ERR_GET_PARTITION_NULL 48
4.5.1.25 QAPI_FW_UPGRADE_ERR_REACH_MAX_IMAGE_ENTRY 48
4.5.1.26 QAPI_FW_UPGRADE_ERR_IMAGE_UNUSED 48
4.5.1.27 QAPI_FW_UPGRADE_ERR_IMAGE_NOT_FOUND 48
4.5.1.28 QAPI_FW_UPGRADE_ERR_IMAGE_DOWNLOAD_FAIL 48
4.5.1.29 QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_CHECKSUM . . . 48
4.5.1.30 QAPI_FW_UPGRADE_ERR_SERVER_RSP_TIMEOUT 48
4.5.1.31 QAPI_FW_UPGRADE_ERR_INVALID_FILENAME 48
4.5.1.32 QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_HDR 48
4.5.1.33 QAPI_FW_UPGRADE_ERR_INSUFFICIENT_MEMORY 48
4.5.1.34 QAPI_FW_UPGRADE_ERR_INCORRECT_SIGNATURE 48
4.5.1.35 QAPI_FW_UPGRADE_ERR_INCORRCT_VERSION 48
4.5.1.36 QAPI_FW_UPGRADE_ERR_INCORRECT_NUM_IMAGES 49
4.5.1.37 QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_LENGTH 49
4.5.1.38 QAPI_FW_UPGRADE_ERR_INCORRECT_HASH_TYPE 49
4.5.1.39 QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_ID 49
4.5.1.40 QAPI_FW_UPGRADE_ERR_SBL_ONLY_NOT_SUPPORT 49
4.5.1.41 QAPI_FW_UPGRADE_ERR_SBL_NOT_SUPPORT_UPGRADE . . . 49
4.5.1.42 QAPI_FW_UPGRADE_ERR_SBL_NOT_ENOUGH_SPACE 49
4.5.1.43 QAPI_FW_UPGRADE_ERR_INVALID_FDT 49
4.5.1.44 QAPI_FW_UPGRADE_ERR_BATTERY_LEVEL_TOO_LOW 49

4.5.1.45 QAPI_FW_UPGRADE_ERR_CRYPTO_FAIL . . . 49
4.5.1.46 QAPI_FW_UPGRADE_ERR_PLUGIN_ENTRY_EMPTY . . . 49
4.5.1.47 QAPI_FW_UPGRADE_ERR_TRIAL_IS_RUNNING . . . 49
4.5.1.48 QAPI_FW_UPGRADE_ERR_FILE_NOT_FOUND . . . 50
4.5.1.49 QAPI_FW_UPGRADE_ERR_FILE_OPEN_ERROR . . . 50
4.5.1.50 QAPI_FW_UPGRADE_ERR_FILE_NAME_TOO_LONG . . . 50
4.5.1.51 QAPI_FW_UPGRADE_ERR_FILE_WRITE_ERROR . . . 50
4.5.1.52 QAPI_FW_UPGRADE_ERR_MOUNT_FILE_SYSTEM_ERROR . . . 50
4.5.1.53 QAPI_FW_UPGRADE_ERR_CREATE_THREAD_ERROR . . . 50
4.5.1.54 QAPI_FW_UPGRADE_ERR_PRESERVE_LAST_FAILED . . . 50
5.1 Flash APIs . . . 51

5 Flash module 51
5.1.1 Define Documentation . . . 51
5.1.1.1 QAPI_FLASH_DEVICE_FAIL . . . 51
5.1.1.2 QAPI_FLASH_DEVICE_NOT_SUPPORTED . . . 51
5.1.1.3 QAPI_FLASH_DEVICE_INVALID_PARAMETER . . . 52
5.1.1.4 QAPI_FLASH_DEVICE_IMAGE_NOT_FOUND . . . 52
5.1.1.5 QAPI_FLASH_DEVICE_NOT_FOUND . . . 52
5.1.1.6 QAPI_FLASH_DEVICE_PENDING . . . 52
5.1.1.7 QAPI_FLASH_DEVICE_NO_MEMORY . . . 52
5.1.1.8 QAPI_FLASH_DEVICE_BUSY . . . 52
5.1.2 Data Structure Documentation . . . 52
5.1.2.1 struct flash_info_s . . . 52
5.1.3 Typedef Documentation . . . 52
5.1.3.1 flash_info_t . . . 52
5.1.4 Enumeration Type Documentation . . . 52
5.1.4.1 qapi_FLASH_Erase_Type_t . . . 52
5.1.5 Function Documentation . . . 53
5.1.5.1 qapi_Flash_Init . . . 53
5.1.5.2 qapi_Flash_Read . . . 53
5.1.5.3 qapi_Flash_Write . . . 53
5.1.5.4 qapi_Flash_Erase . . . 54
5.1.5.5 qapi_Flash_Get_Info . . . 54
5.2 Flash config APIs . . . 55
6.1 Heap status APIs . . . 56

6 Heap Status module 56
6.1.1 Data Structure Documentation . . . 56
6.1.1.1 struct heap_status_t . . . 56
6.1.2 Function Documentation . . . 56
6.1.2.1 qapi_Heap_Status . . . 56

7 I2C module 57
7.1 I2C APIs . . . 58
7.1.1 Data Structure Documentation . . . 58
7.1.1.1 struct qapi_I2CM_Config_s . . . 58
7.1.1.2 struct qapi_I2CM_Transfer_Config_s . . . 58
7.1.1.3 struct qapi_I2CM_Descriptor_s . . . 58
7.1.2 Typedef Documentation . . . 58

7.1.2.1 qapi_I2CM_Config_t . . . 58
7.1.2.2 qapi_I2CM_Transfer_Config_t . . . 59
7.1.2.3 qapi_I2CM_Descriptor_t . . . 59
7.1.2.4 qapi_I2CM_Transfer_CB_t . . . 59
7.1.3 Enumeration Type Documentation . . . 59
7.1.3.1 qapi_I2CM_Instance_t . . . 59
7.1.4 Function Documentation . . . 59
7.1.4.1 qapi_I2CM_Open . . . 59
7.1.4.2 qapi_I2CM_Close . . . 60
7.1.4.3 qapi_I2CM_Transfer . . . 60
7.1.4.4 qapi_I2CM_Cancel_Transfer . . . 61
7.2 I2C error codes . . . 62
7.2.1 Define Documentation . . . 62
7.2.1.1 QAPI_I2CM_ERROR . . . 62
7.2.1.2 QAPI_I2CM_ERROR_INVALID_PARAM . . . 62
7.2.1.3 QAPI_I2CM_ERROR_MEM_ALLOC . . . 62
7.2.1.4 QAPI_I2CM_ERROR_TRANSFER_BUSY . . . 62
7.2.1.5 QAPI_I2CM_ERROR_TRANSFER_TIMEOUT . . . 62
7.2.1.6 QAPI_I2CM_ERROR_INPUT_FIFO_UNDER_RUN . . . 62
7.2.1.7 QAPI_I2CM_ERROR_INPUT_FIFO_OVER_RUN . . . 62
7.2.1.8 QAPI_I2CM_ERROR_OUTPUT_FIFO_UNDER_RUN . . . 62
7.2.1.9 QAPI_I2CM_ERROR_OUTPUT_FIFO_OVER_RUN . . . 62
7.2.1.10 QAPI_I2CM_ERROR_COMMAND_OVER_RUN . . . 62
7.2.1.11 QAPI_I2CM_ERROR_TRANSFER_FORCE_TERMINATED . . . 62
7.2.1.12 QAPI_I2CM_ERROR_COMMAND_ILLEGAL . . . 63
7.2.1.13 QAPI_I2CM_ERROR_COMMAND_FAIL . . . 63
7.2.1.14 QAPI_I2CM_ERROR_BUS_CLK_ENABLE_FAIL . . . 63
7.2.1.15 QAPI_I2CM_ERROR_BUS_GPIO_ENABLE_FAIL . . . 63
7.2.1.16 QAPI_I2CM_ERROR_DMA_TX_BUS_ERROR . . . 63
7.2.1.17 QAPI_I2CM_ERROR_DMA_RX_BUS_ERROR . . . 63
7.2.1.18 QAPI_I2CM_DMA_TX_RESET_DONE . . . 63
7.2.1.19 QAPI_I2CM_DMA_RX_RESET_DONE . . . 63
7.2.1.20 QAPI_I2CM_CANCEL_TRANSFER_COMPLETED . . . 63
7.2.1.21 QAPI_I2CM_CANCEL_TRANSFER_INVALID . . . 63
7.2.1.22 QAPI_I2CM_ERROR_CANCEL_TRANSFER_FAIL . . . 63
7.2.1.23 QAPI_I2CM_ERROR_BOOTSTRAP_CFG_FAIL . . . 63
7.2.1.24 QAPI_I2CM_ERROR_DEVICE_STATE . . . 64
7.2.1.25 QAPI_I2CM_ERROR_INIT_XFR . . . 64
7.2.1.26 QAPI_I2CM_ERROR_IC_COMP . . . 64
7.2.1.27 QAPI_I2CM_ERROR_TX_ABORT_INTR . . . 64
7.3 I2C transfer flags . . . 65
7.3.1 Define Documentation . . . 65
7.3.1.1 QAPI_I2C_FLAG_START . . . 65
7.3.1.2 QAPI_I2C_FLAG_STOP . . . 65
7.3.1.3 QAPI_I2C_FLAG_WRITE . . . 65
7.3.1.4 QAPI_I2C_FLAG_READ . . . 65
8.1 Low power APIs . . . 66

8 Low Power module 66
8.1.1 Typedef Documentation . . . 66

8.1.1.1 qapi_bmps_rx_filter_cb . . . 66
8.1.2 Function Documentation . . . 66
8.1.2.1 qapi_pm_enable . . . 66
8.1.2.2 qapi_deepsleep_enter . . . 66
8.1.2.3 qapi_imps_enter_sleep . . . 67
8.1.2.4 qapi_imps_disable_sleep . . . 67
8.1.2.5 qapi_bmps_cfg . . . 67
8.1.2.6 qapi_bmps_rx_filter_enable . . . 68
8.1.2.7 qapi_bmps_bcmc_rx_filter_cb_register . . . 68
8.1.2.8 qapi_bmps_sleep_wakeup_cb . . . 69
8.1.2.9 qapi_bmps_get_exit_reason . . . 69
9.1 WLAN . . . 70

9 WLAN module 70
9.1.1 Define Documentation . . . 70
9.1.1.1 __QAPI_WLAN_DEFAULT_SCAN_CTRL_FLAGS . . . 70
9.1.1.2 __QAPI_WLAN_PMKID_LEN . . . 70
9.1.1.3 __QAPI_WLAN_DBGLOG_MISC_CONFIG_FLUSH_FLAG_BIT_OFFSET . . . 70
9.1.1.4 __QAPI_WLAN_DBGLOG_MISC_CONFIG_FLUSH_FLAG_MASK . . . 70
9.1.1.5 QAPI_TX_POWER_RESTORE . . . 70
9.1.1.6 MIN_GEN_PKT_INTERVAL . . . 70
9.1.2 Data Structure Documentation . . . 70
9.1.2.1 struct qapi_WLAN_Start_Scan_Params_t . . . 70
9.1.2.2 struct qapi_WLAN_BSS_Scan_Info_t . . . 71
9.1.2.3 struct qapi_WLAN_BSS_Scan_Info_ExtV1_t . . . 71
9.1.2.4 struct qapi_WLAN_Evt_Hdr_t . . . 72
9.1.2.5 struct qapi_WLAN_Enable_Evt_t . . . 72
9.1.2.6 struct qapi_WLAN_Disable_Evt_t . . . 72
9.1.2.7 struct qapi_WLAN_If_Add_Comp_Evt_t . . . 73
9.1.2.8 struct qapi_WLAN_Scan_Start_Evt_t . . . 73
9.1.2.9 struct qapi_WLAN_Scan_Comp_Evt_t . . . 73
9.1.2.10 struct qapi_WLAN_Join_Comp_Evt_t . . . 73
9.1.2.11 struct qapi_WLAN_Chan_Switch_Evt_t . . . 74
9.1.2.12 struct qapi_WLAN_WPS_Fail_Evt_t . . . 74
9.1.2.13 struct qapi_WLAN_Connect_Cb_Info_t . . . 75
9.1.2.14 struct qapi_WLAN_Connect_Cb_Info_ExtV1_t . . . 76
9.1.2.15 struct qapi_WLAN_Ready_Cb_Info_t . . . 76
9.1.2.16 struct qapi_WLAN_Device_Stats_t . . . 76
9.1.2.17 struct qapi_WLAN_Common_TxRx_Buffer_Info . . . 79
9.1.2.18 struct qapi_WLAN_Common_Stats_t . . . 79
9.1.2.19 struct qapi_WLAN_Device_Stats_Ext_t . . . 80
9.1.2.20 struct qapi_WLAN_Device_Stats_Ext2_t . . . 80
9.1.2.21 struct qapi_WLAN_Statistics_t . . . 80
9.1.2.22 struct qapi_WLAN_Get_Statistics_t . . . 81
9.1.2.23 struct qapi_WLAN_Get_Channel_List_t . . . 81
9.1.2.24 struct qapi_WLAN_Preferred_Network_Offload_Info_t . . . 82
9.1.2.25 struct qapi_WLAN_Event_Filter_t . . . 82
9.1.2.26 struct qapi_WLAN_Firmware_Version_String_t . . . 82
9.1.2.27 struct qapi_WLAN_Scan_Params_t . . . 83

9.1.2.28 struct qapi_WLAN_Power_Mode_Params_t . . . 84
9.1.2.29 struct qapi_WLAN_Power_Policy_Params_t . . . 85
9.1.2.30 struct qapi_WLAN_Rssi_Threshold_Params_t . . . 85
9.1.2.31 struct qapi_WLAN_LowRssi_RoamControl_Params_t . . . 86
9.1.2.32 struct qapi_WLAN_Channel_Switch_t . . . 86
9.1.2.33 struct qapi_WLAN_TCP_Offload_Enable_t . . . 87
9.1.2.34 struct qapi_WLAN_TCP_Offload_Config_Params_t . . . 87
9.1.2.35 struct qapi_WLAN_TCP_Keepalive_Event_Info_t . . . 88
9.1.2.36 struct qapi_WLAN_TCP_Keepalive_Event_t . . . 89
9.1.2.37 struct qapi_WLAN_A_Netbuf_Pool_Config_t . . . 89
9.1.2.38 struct qapi_WLAN_Add_Pattern_t . . . 89
9.1.2.39 struct qapi_WLAN_Delete_Pattern_t . . . 90
9.1.2.40 struct qapi_WLAN_Change_Default_Filter_Action_t . . . 90
9.1.2.41 struct qapi_WLAN_Security_Wep_Key_Pair_Params_t . . . 90
9.1.2.42 struct qapi_WLAN_Security_8021x_Private_Key_t . . . 91
9.1.2.43 struct qapi_WLAN_Preferred_Network_Offload_Config_t . . . 91
9.1.2.44 struct qapi_WLAN_Preferred_Network_Profile_t . . . 92
9.1.2.45 struct qapi_WLAN_Promiscuous_Mode_Info_t . . . 92
9.1.2.46 struct qapi_WLAN_Raw_Send_Params_t . . . 93
9.1.2.47 struct qapi_WLAN_Set_PMKID_Params_t . . . 94
9.1.2.48 struct qapi_WLAN_RxEapolKey_Cb_Info_t . . . 94
9.1.2.49 struct qapi_WLAN_Set_Finite_Cyclic_Groups_Params_t . . . 95
9.1.2.50 struct qapi_WLAN_SAE_PMK_Cache_t . . . 95
9.1.2.51 struct qapi_WLAN_WPS_Credentials_t . . . 95
9.1.2.52 struct qapi_WLAN_WPS_Cb_Info_t . . . 96
9.1.2.53 struct qapi_WLAN_Aggregation_Params_t . . . 96
9.1.2.54 struct qapi_WLAN_Scan_List_t . . . 96
9.1.2.55 struct qapi_WLAN_5G_Prefer_t . . . 97
9.1.2.56 struct qapi_WLAN_Connect_Policy_t . . . 97
9.1.2.57 struct qapi_WLAN_Set_BMcast_Filter_t . . . 97
9.1.2.58 struct qapi_WLAN_Tx_Status_t . . . 97
9.1.2.59 struct qapi_WLAN_Program_Mac_Addr_Param_t . . . 98
9.1.2.60 struct qapi_WLAN_Dbglog_Enable_t . . . 98
9.1.2.61 struct qapi_WLAN_Dbglog_Module_Config_t . . . 98
9.1.2.62 struct qapi_WLAN_Dbglog_Config_t . . . 99
9.1.2.63 struct qapi_WPS_Scan_List_Entry_t . . . 99
9.1.2.64 struct qapi_WLAN_WPS_Start_t . . . 100
9.1.2.65 struct qapi_WLAN_Cipher_t . . . 100
9.1.2.66 struct qapi_WLAN_Netparams_t . . . 100
9.1.2.67 union qapi_WLAN_Netparams_t.u . . . 102
9.1.2.68 struct qapi_WLAN_IPv6_Addr_t . . . 102
9.1.2.69 struct qapi_WLAN_ARP_Offload_Config_t . . . 102
9.1.2.70 struct qapi_WLAN_NS_Offload_Config_t . . . 102
9.1.2.71 struct qapi_WLAN_App_Ie_Params_t . . . 103
9.1.2.72 struct qapi_WLAN_Set_Txpower_Params_t . . . 103
9.1.2.73 struct qapi_WLAN_Rx_Aggrx_Params_t . . . 103
9.1.2.74 struct qapi_WLAN_BSS_Max_Idle_Period_t . . . 104
9.1.2.75 struct qapi_WLAN_WNM_Sleep_Period_t . . . 104
9.1.2.76 struct qapi_WLAN_WNM_Cb_Info_t . . . 104
9.1.2.77 struct qapi_WLAN_Sta_Config_Bmiss_Config_t . . . 104

9.1.2.78 struct qapi_WLAN_Hw_Ant_Div_Config_t . . . 105
9.1.2.79 struct qapi_WLAN_Ant_Div_Config_t . . . 105
9.1.2.80 union qapi_WLAN_Ant_Div_Config_t.ant_div_config . . . 105
9.1.2.81 struct qapi_WLAN_Get_Ant_Div_t . . . 106
9.1.2.82 struct qapi_WLAN_Pmf_Evt_t . . . 106
9.1.2.83 struct qapi_WLAN_Gen_Pkt_Params_t . . . 106
9.1.2.84 struct qapi_WLAN_Reg_t . . . 107
9.1.2.85 struct qapi_WLAN_Reg_Evt_t . . . 107
9.1.2.86 struct qapi_WLAN_Get_Power_Evt_t . . . 107
9.1.2.87 struct qapi_WLAN_Set_Rate_Params_t . . . 107
9.1.2.88 struct qapi_WLAN_Listen_Interval_Params_t . . . 108
9.1.2.89 struct qapi_WLAN_Edca_Params_t . . . 108
9.1.2.90 struct qapi_WLAN_BA_Window_Params_t . . . 108
9.1.3 Typedef Documentation . . . 108
9.1.3.1 qapi_WLAN_Callback_t . . . 108
9.1.4 Enumeration Type Documentation . . . 109
9.1.4.1 qapi_WLAN_Enable_e . . . 109
9.1.4.2 qapi_WLAN_Scan_Ctrl_Flag_bits_e . . . 109
9.1.4.3 qapi_WLAN_Scan_Status_e . . . 109
9.1.4.4 qapi_WLAN_Callback_ID_e . . . 109
9.1.4.5 qapi_WLAN_Filterable_Event_e . . . 111
9.1.4.6 qapi_WLAN_Wait_For_Status_e . . . 112
9.1.4.7 qapi_WLAN_Dev_Mode_e . . . 112
9.1.4.8 qapi_WLAN_Phy_Mode_e . . . 112
9.1.4.9 qapi_WLAN_11n_HT_Config_e . . . 112
9.1.4.10 qapi_WLAN_Power_Mode_e . . . 113
9.1.4.11 qapi_WLAN_Power_Module_e . . . 113
9.1.4.12 qapi_WLAN_DTIM_Policy_e . . . 113
9.1.4.13 qapi_WLAN_TX_Wakeup_Policy_e . . . 114
9.1.4.14 qapi_WLAN_Power_Save_Policy_e . . . 114
9.1.4.15 qapi_WLAN_IP_Protocol_Type_e . . . 114
9.1.4.16 qapi_WLAN_Pkt_Filter_Type_e . . . 114
9.1.4.17 qapi_WLAN_Auth_Mode_e . . . 115
9.1.4.18 qapi_WLAN_Crypt_Type_e . . . 116
9.1.4.19 qapi_WLAN_8021X_Method_e . . . 116
9.1.4.20 qapi_WLAN_Raw_Mode_Header_Type_e . . . 116
9.1.4.21 qapi_WLAN_PMKID_ENABLE_e . . . 117
9.1.4.22 qapi_EAPOL_KEY_TYPE_e . . . 117
9.1.4.23 qapi_WLAN_WPS_Connect_Action_e . . . 117
9.1.4.24 qapi_WLAN_WPS_Mode_e . . . 117
9.1.4.25 qapi_WLAN_WPS_Status_Code_e . . . 117
9.1.4.26 qapi_WLAN_WPS_Error_Code_e . . . 118
9.1.4.27 qapi_WLAN_Mac_Result_e . . . 118
9.1.4.28 qapi_WLAN_Dbglog_Port_e . . . 118
9.1.4.29 qapi_WLAN_Mode_e . . . 118
9.1.4.30 qapi_WLAN_DEV_Mode_e . . . 119
9.1.4.31 qapi_WLAN_Mgmt_Frame_Type_e . . . 119
9.1.4.32 qapi_WLAN_TX_Power_Policy_e . . . 119
9.1.4.33 qapi_WLAN_Disconnect_Reason_t . . . 119
9.1.4.34 qapi_WLAN_Ant_Div_Mode_e . . . 120

9.1.4.35 qapi_WLAN_Ant_Div_Enable_e . . . 120
9.1.4.36 qapi_WLAN_PMF_EVENT_TYPE_e . . . 120
9.1.4.37 qapi_WLAN_PMF_CAP_e . . . 120
9.1.4.38 qapi_WLAN_RSN_CAP_e . . . 120
9.1.4.39 qapi_WLAN_Wpa3_Enable_e . . . 121
9.1.4.40 qapi_WLAN_MGMT_FRAME_e . . . 121
9.1.4.41 qapi_WLAN_FTM_CMD_e . . . 121
9.1.5 Function Documentation . . . 121
9.1.5.1 qapi_WLAN_Enable . . . 121
9.1.5.2 qapi_WLAN_Add_Device . . . 122
9.1.5.3 qapi_WLAN_Start_Scan . . . 122
9.1.5.4 qapi_WLAN_Get_Scan_Results . . . 123
9.1.5.5 qapi_WLAN_Commit . . . 123
9.1.5.6 qapi_WLAN_Set_Callback . . . 124
9.1.5.7 qapi_WLAN_Disconnect . . . 125
9.1.5.8 qapi_WLAN_Set_Param . . . 125
9.1.5.9 qapi_WLAN_Get_Param . . . 126
9.1.5.10 qapi_WLAN_Set_11d . . . 126
9.1.5.11 qapi_WLAN_Get_11d . . . 127
9.1.5.12 qapi_WLAN_Get_Country_Code . . . 127
9.1.5.13 qapi_WLAN_Get_Regulatory_Info . . . 127
9.1.5.14 qapi_WLAN_Set_Rate . . . 128
9.1.5.15 qapi_WLAN_Get_Rate . . . 128
9.1.5.16 qapi_WLAN_Raw_Send . . . 128
9.1.5.17 qapi_WLAN_Enable_Mgmt_Filter . . . 129
9.1.5.18 qapi_WLAN_Disable_Mgmt_Filter . . . 129
9.1.5.19 qapi_WLAN_Recv_Mgmt_Frames . . . 129
9.2 WLAN errors . . . 130
9.2.1 Define Documentation . . . 130
9.2.1.1 QAPI_WLAN_ERROR . . . 130
9.2.1.2 QAPI_WLAN_ERR_DEVICE_NOT_FOUND . . . 130
9.2.1.3 QAPI_WLAN_ERR_NO_MEMORY . . . 130
9.2.1.4 QAPI_WLAN_ERR_MEMORY_NOT_AVAIL . . . 130
9.2.1.5 QAPI_WLAN_ERR_NO_FREE_DESC . . . 130
9.2.1.6 QAPI_WLAN_ERR_BAD_ADDRESS . . . 130
9.2.1.7 QAPI_WLAN_ERR_WIN_DRIVER_ERROR . . . 130
9.2.1.8 QAPI_WLAN_ERR_REGS_NOT_MAPPED . . . 130
9.2.1.9 QAPI_WLAN_ERR_EPERM . . . 130
9.2.1.10 QAPI_WLAN_ERR_EACCES . . . 130
9.2.1.11 QAPI_WLAN_ERR_ENOENT . . . 130
9.2.1.12 QAPI_WLAN_ERR_EEXIST . . . 131
9.2.1.13 QAPI_WLAN_ERR_EFAULT . . . 131
9.2.1.14 QAPI_WLAN_ERR_EBUSY . . . 131
9.2.1.15 QAPI_WLAN_ERR_EINVAL . . . 131
9.2.1.16 QAPI_WLAN_ERR_EMSGSIZE . . . 131
9.2.1.17 QAPI_WLAN_ERR_ECANCELED . . . 131
9.2.1.18 QAPI_WLAN_ERR_ENOTSUP . . . 131
9.2.1.19 QAPI_WLAN_ERR_ECOMM . . . 131
9.2.1.20 QAPI_WLAN_ERR_EPROTO . . . 131
9.2.1.21 QAPI_WLAN_ERR_ENODEV . . . 131

9.2.1.22 QAPI_WLAN_ERR_EDEVNOTUP . . . 131
9.2.1.23 QAPI_WLAN_ERR_NO_RESOURCE . . . 131
9.2.1.24 QAPI_WLAN_ERR_HARDWARE . . . 132
9.2.1.25 QAPI_WLAN_ERR_PENDING . . . 132
9.2.1.26 QAPI_WLAN_ERR_EBADCHANNEL . . . 132
9.2.1.27 QAPI_WLAN_ERR_DECRYPT_ERROR . . . 132
9.2.1.28 QAPI_WLAN_ERR_PHY_ERROR . . . 132
9.2.1.29 QAPI_WLAN_ERR_CONSUMED . . . 132
9.2.1.30 QAPI_WLAN_ERR_CLONE . . . 132
9.2.1.31 QAPI_WLAN_ERR_HW_CONFIG_ERROR . . . 132
9.2.1.32 QAPI_WLAN_ERR_SOCKCXT_NOT_FOUND . . . 132
9.2.1.33 QAPI_WLAN_ERR_UNKNOWN_CMD . . . 132
9.2.1.34 QAPI_WLAN_ERR_SOCK_UNAVAILABLE . . . 132
9.2.1.35 QAPI_WLAN_ERR_MEMFREE_ERROR . . . 132
9.2.1.36 QAPI_WLAN_ERR_BUS_ERROR . . . 133
9.2.1.37 QAPI_WLAN_ERR_QOSAL_INVALID_TASK_ID . . . 133
9.2.1.38 QAPI_WLAN_ERR_QOSAL_INVALID_PARAMETER . . . 133
9.2.1.39 QAPI_WLAN_ERR_QOSAL_INVALID_POINTER . . . 133
9.2.1.40 QAPI_WLAN_ERR_QOSAL_ALREADY_EXISTS . . . 133
9.2.1.41 QAPI_WLAN_ERR_QOSAL_INVALID_EVENT . . . 133
9.2.1.42 QAPI_WLAN_ERR_QOSAL_EVENT_TIMEOUT . . . 133
9.2.1.43 QAPI_WLAN_ERR_QOSAL_INVALID_MUTEX . . . 133
9.2.1.44 QAPI_WLAN_ERR_QOSAL_TASK_ALREADY_LOCKED . . . 133
9.2.1.45 QAPI_WLAN_ERR_QOSAL_MUTEX_ALREADY_LOCKED . . . 133
9.2.1.46 QAPI_WLAN_ERR_QOSAL_OUT_OF_MEMORY . . . 133
9.2.1.47 QAPI_WLAN_ERR_IMG_VERIFY_ERROR . . . 133
9.2.1.48 QAPI_WLAN_ERR_FLASH_ERROR . . . 134
9.3 WLAN control operations . . . 135
9.3.1 Define Documentation . . . 135
9.3.1.1 __QAPI_WLAN_DEVICE0_NAME . . . 135
9.3.1.2 __QAPI_WLAN_DEVICE1_NAME . . . 135
9.3.1.3 __QAPI_WLAN_MAC_LEN . . . 135
9.3.1.4 __QAPI_WLAN_MAX_SSID_LEN . . . 135
9.3.1.5 __QAPI_WLAN_MAX_SSID_LENGTH . . . 135
9.3.1.6 __QAPI_WLAN_PASSPHRASE_LEN . . . 135
9.3.1.7 __QAPI_WLAN_PNO_MAX_NETWORK_PROFILES . . . 135
9.3.1.8 __QAPI_WLAN_IPV6_ADDR_LEN . . . 135
9.3.1.9 __QAPI_WLAN_IPV4_ADDR_LEN . . . 135
9.3.1.10 __QAPI_WLAN_WPS_PIN_LEN . . . 135
9.3.1.11 __QAPI_WLAN_WPS_MAX_KEY_LEN . . . 135
9.3.1.12 __QAPI_WLAN_PROMISC_MAX_FILTER_IDX . . . 135
9.3.1.13 __QAPI_WLAN_MAX_NUM_CUR_REGDOAMIN_CHANLIST_CHN-NELS . . . 135
9.3.1.14 __QAPI_WLAN_PROM_FILTER_SOURCE . . . 136
9.3.1.15 __QAPI_WLAN_PROM_FILTER_DEST . . . 136
9.3.1.16 __QAPI_WLAN_PROM_FILTER_FRAME_TYPE . . . 136
9.3.1.17 __QAPI_WLAN_PROM_FILTER_FRAME_SUB_TYPE . . . 137
9.3.1.18 __QAPI_WLAN_PROMISC_REPORT_RSSI_INFO . . . 137
9.3.1.19 __QAPI_WLAN_PROMISC_INDICATE_APPEND_PAYLOAD . . . 137
9.3.1.20 __QAPI_WLAN_DBGLOG_MAX_MODULE_ID . . . 137

9.3.1.21 __QAPI_WLAN_DBGLOG_LOGLEVEL_INFO_LEN 137
9.3.1.22 __QAPI_WLAN_DBGLOG_DEFAULT_MODULE_MASK 137
9.3.1.23 __QAPI_WLAN_DBGLOG_GLOBAL_MODULE_ID 137
9.3.1.24 __QAPI_WLAN_DBGLOG_DEFAULT_MODULE_LOG_LEVEL 137
9.3.1.25 __QAPI_WLAN_DBGLOG_LOCAL_DBG_PORT 137
9.3.1.26 __QAPI_WLAN_DBGLOG_REMOTE_DBG_PORT 138
9.3.1.27 __QAPI_WLAN_DBGLOG_CFG_LOG_LEVEL_FLAG_MASK 138
9.3.1.28 __QAPI_WLAN_DBGLOG_CFG_DEBUG_PORT_FLAG_MASK 138
9.3.1.29 __QAPI_WLAN_DBGLOG_CFG_REPORT_FLAG_MASK 138
9.3.1.30 __QAPI_WLAN_DBGLOG_CFG_REPORT_SIZE_FLAG_MASK 138
9.3.1.31 __QAPI_WLAN_DBGLOG_CFG_TIMER_RESOLUTION_FLAG_MASK 138
9.3.1.32 __QAPI_WLAN_DBGLOG_CFG_FLUSH_FLAG_MASK 138
9.3.1.33 __QAPI_WLAN_DBGLOG_NIBBLE_CNT_IN_WORD_MEMORY . . . 138
9.3.1.34 __QAPI_WLAN_DBGLOG_BIT_CNT_IN_WORD_MEMORY 138
9.3.1.35 __QAPI_WLAN_CHAN_FREQ_1 138
9.3.1.36 __QAPI_WLAN_CHAN_FREQ_2 138
9.3.1.37 __QAPI_WLAN_CHAN_FREQ_3 138
9.3.1.38 __QAPI_WLAN_CHAN_FREQ_4 138
9.3.1.39 __QAPI_WLAN_CHAN_FREQ_5 138
9.3.1.40 __QAPI_WLAN_CHAN_FREQ_6 139
9.3.1.41 __QAPI_WLAN_CHAN_FREQ_7 139
9.3.1.42 __QAPI_WLAN_CHAN_FREQ_8 139
9.3.1.43 __QAPI_WLAN_CHAN_FREQ_9 139
9.3.1.44 __QAPI_WLAN_CHAN_FREQ_10 139
9.3.1.45 __QAPI_WLAN_CHAN_FREQ_11 139
9.3.1.46 __QAPI_WLAN_CHAN_FREQ_12 139
9.3.1.47 __QAPI_WLAN_CHAN_FREQ_13 139
9.3.1.48 __QAPI_WLAN_CHAN_FREQ_14 139
9.3.1.49 __QAPI_WLAN_CHAN_FREQ_36 139
9.3.1.50 __QAPI_WLAN_CHAN_FREQ_40 139
9.3.1.51 __QAPI_WLAN_CHAN_FREQ_44 139
9.3.1.52 __QAPI_WLAN_CHAN_FREQ_48 139
9.3.1.53 __QAPI_WLAN_CHAN_FREQ_52 139
9.3.1.54 __QAPI_WLAN_CHAN_FREQ_56 139
9.3.1.55 __QAPI_WLAN_CHAN_FREQ_60 139
9.3.1.56 __QAPI_WLAN_CHAN_FREQ_64 140
9.3.1.57 __QAPI_WLAN_CHAN_FREQ_100 140
9.3.1.58 __QAPI_WLAN_CHAN_FREQ_104 140
9.3.1.59 __QAPI_WLAN_CHAN_FREQ_108 140
9.3.1.60 __QAPI_WLAN_CHAN_FREQ_112 140
9.3.1.61 __QAPI_WLAN_CHAN_FREQ_116 140
9.3.1.62 __QAPI_WLAN_CHAN_FREQ_132 140
9.3.1.63 __QAPI_WLAN_CHAN_FREQ_136 140
9.3.1.64 __QAPI_WLAN_CHAN_FREQ_140 140
9.3.1.65 __QAPI_WLAN_CHAN_FREQ_149 140
9.3.1.66 __QAPI_WLAN_CHAN_FREQ_153 140
9.3.1.67 __QAPI_WLAN_CHAN_FREQ_157 140
9.3.1.68 __QAPI_WLAN_CHAN_FREQ_161 140
9.3.1.69 __QAPI_WLAN_CHAN_FREQ_165 140

9.3.1.70 __QAPI_WLAN_6G_CHAN_FREQ_1 . . . 140
9.3.1.71 __QAPI_WLAN_SECURITY_AUTH_PSK . . . 140
9.3.1.72 __QAPI_WLAN_SECURITY_AUTH_1X . . . 141
9.3.1.73 __QAPI_WLAN_SECURITY_AUTH_SAE . . . 141
9.3.1.74 __QAPI_WLAN_CIPHER_TYPE_TKIP . . . 141
9.3.1.75 __QAPI_WLAN_CIPHER_TYPE_CCMP . . . 141
9.3.1.76 __QAPI_WLAN_CIPHER_TYPE_WEP . . . 141
9.3.1.77 __QAPI_WLAN_MAX_WEP_KEY_SZ . . . 141
9.3.1.78 __QAPI_MAX_SCAN_RESULT_ENTRY . . . 141
9.3.1.79 __QAPI_WLAN_SHORTSCANRATIO_DEFAULT . . . 141
9.3.1.80 __QAPI_WLAN_MAX_NS_OFFLOAD_TUPLES . . . 141
9.3.1.81 __QAPI_WLAN_MAX_ARP_OFFLOAD_TUPLES . . . 141
9.3.1.82 __QAPI_WLAN_NSOFF_MAX_TARGET_IPS . . . 141
9.3.1.83 __QAPI_WLAN_START_SCAN_PARAMS_CHANNEL_LIST_MAX . . 142
9.3.1.84 __QAPI_WLAN_MAX_NUM_FILTERED_EVENTS . . . 142
9.3.1.85 __QAPI_WLAN_VERSION_SUBSTRING_LEN . . . 142
9.3.1.86 __QAPI_WLAN_PATTERN_MAX_SIZE . . . 142
9.3.1.87 __QAPI_WLAN_PATTERN_MASK . . . 142
9.3.1.88 __QAPI_WLAN_PATTERN_REJECT_FLAG . . . 142
9.3.1.89 __QAPI_WLAN_PATTERN_ACCEPT_FLAG . . . 142
9.3.1.90 __QAPI_WLAN_PATTERN_DEFER_FLAG . . . 143
9.3.1.91 __QAPI_WLAN_PATTERN_WOW_FLAG . . . 143
9.3.1.92 __QAPI_WLAN_DEFAULT_FILTER_INDEX . . . 143
9.3.1.93 __QAPI_WLAN_MAX_FILTER_INDEX . . . 143
9.3.1.94 __QAPI_WLAN_DELETE_ALL_FILTER_INDEX . . . 143
9.3.1.95 __QAPI_WLAN_DEFAULT_FILTER_PRIORITY . . . 143
9.3.1.96 __QAPI_WLAN_TX_STATUS_IDLE . . . 143
9.3.1.97 __QAPI_WLAN_TX_STATUS_HOST_PENDING . . . 143
9.3.1.98 __QAPI_WLAN_TX_STATUS_WIFI_PENDING . . . 143
9.3.1.99 __QAPI_WLAN_AGGRX_BUFFER_SIZE_MAX . . . 144
9.3.1.100 __QAPI_WLAN_AGGRX_CFG_INVAL . . . 144
9.3.2 Enumeration Type Documentation . . . 144
9.3.2.1 qapi_WLAN_Bit_Rate_t . . . 144
9.4 WLAN parameter group information . . . 146
9.4.1 Define Documentation . . . 146
9.4.1.1 __QAPI_WLAN_PARAM_GROUP_SYSTEM . . . 146
9.4.1.2 __QAPI_WLAN_PARAM_GROUP_WIRELESS . . . 146
9.4.1.3 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SECURITY . . . 146
9.4.1.4 __QAPI_WLAN_PARAM_GROUP_P2P . . . 146
9.4.1.5 __QAPI_WLAN_PARAM_GROUP_SYSTEM_ENABLE_SUSPEND_-RESUME . . . 146
9.4.1.6 __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_ENABLE . . . 146
9.4.1.7 __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_CONFIG . . . 146
9.4.1.8 __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_MODULE_-CONFIG . . . 146
9.4.1.9 __QAPI_WLAN_PARAM_GROUP_SYSTEM_FIRMWARE_VERSION 147
9.4.1.10 __QAPI_WLAN_PARAM_GROUP_SYSTEM_DRIVER_NETBUF_P-OOL_SIZE . . . 147
9.4.1.11 __QAPI_WLAN_PARAM_GROUP_SYSTEM_LAST_ERROR . . . 147
9.4.1.12 __QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE 147

9.4.1.13 __QAPI_WLAN_PARAM_GROUP_WIRELESS_CHANNEL 148
9.4.1.14 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SCAN_PARAMS . . . 148
9.4.1.15 __QAPI_WLAN_PARAM_GROUP_WIRELESS_TX_POWER_IN_DBM 148
9.4.1.16 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SSID 148
9.4.1.17 __QAPI_WLAN_PARAM_GROUP_WIRELESS_BSSID 149
9.4.1.18 __QAPI_WLAN_PARAM_GROUP_WIRELESS_PHY_MODE 149
9.4.1.19 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROBE_-REQ_FWD_TO_HOST 149
9.4.1.20 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ALLOW_TX_RX_A-GGR_SET_TID 150
9.4.1.21 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_ROAMING . 150
9.4.1.22 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISC-UOUS_MODE 150
9.4.1.23 __QAPI_WLAN_PARAM_GROUP_WIRELESS_POWER_MODE_P-OLICY 150
9.4.1.24 __QAPI_WLAN_PARAM_GROUP_WIRELESS_POWER_MODE_P-ARAMS 151
9.4.1.25 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_WAKE_O-N_WIRELESS 151
9.4.1.26 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ADD_PATTERN . . . 151
9.4.1.27 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_GREEN_-TX 152
9.4.1.28 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_LOW_PO-WER_LISTEN 152
9.4.1.29 __QAPI_WLAN_PARAM_GROUP_WIRELESS_RATE 152
9.4.1.30 __QAPI_WLAN_PARAM_GROUP_WIRELESS_REG_DOMAIN 153
9.4.1.31 __QAPI_WLAN_PARAM_GROUP_WIRELESS_TX_STATUS 153
9.4.1.32 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STATS 153
9.4.1.33 __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NET-WORK_OFFLOAD_ENABLE 153
9.4.1.34 __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NET-WORK_PROFILE 154
9.4.1.35 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_KEEP_ALIVE_-IN_SEC 154
9.4.1.36 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_LISTEN_INTE-RVAL_IN_TU 154
9.4.1.37 __QAPI_WLAN_PARAM_GROUP_WIRELESS_RSSI 155
9.4.1.38 __QAPI_WLAN_PARAM_GROUP_WIRELESS_RSSI_THRESHOLD . 155
9.4.1.39 __QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_-OFFLOAD_ENABLE 155
9.4.1.40 __QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_-OFFLOAD_SESSION_CFG 155
9.4.1.41 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_UAPSD 156
9.4.1.42 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_MAX_SP_LEN . 156
9.4.1.43 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AMSDU_RX 156
9.4.1.44 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_BEACON_INTE-RVAL_IN_TU 156
9.4.1.45 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_ENABLE_HIDD-EN_MODE 157

9.4.1.46 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_INACTIVITY_TIME_IN_MINS . . . 157
9.4.1.47 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_WPS_FLAG . . . 157
9.4.1.48 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_DTIM_INTERVAL . . . 158
9.4.1.49 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ARP_OFFLOAD_PARAMS . . . 158
9.4.1.50 __QAPI_WLAN_PARAM_GROUP_WIRELESS_NS_OFFLOAD_PARAMS . . . 159
9.4.1.51 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PKT_FILTER . . . 159
9.4.1.52 __QAPI_WLAN_PARAM_GROUP_WIRELESS_DELETE_PATTERN . 159
9.4.1.53 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_ENABLE_UAPSD 160
9.4.1.54 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AGGRX_CONFIG . . 160
9.4.1.55 __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG 160
9.4.1.56 __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_STA_SLEEP_PERIOD . . . 161
9.4.1.57 __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_AP_BSS_MAX_IDLE_PERIOD . . . 161
9.4.1.58 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_CHANNEL_SWITCH . . . 162
9.4.1.59 __QAPI_WLAN_PARAM_GROUP_WIRELESS_EVENT_FILTER . . . 162
9.4.1.60 __QAPI_WLAN_PARAM_GROUP_WIRELESS_11N_HT 162
9.4.1.61 __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_BMISS_CONFIG . . . 163
9.4.1.62 __QAPI_WLAN_PARAM_GROUP_WIRELESS_CHANGE_DEFAULT_FILTER_ACTION . . . 163
9.4.1.63 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_COUNTRY_CODE . . . 163
9.4.1.64 __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_CURR_REGDOMAIN_CHANNEL_LIST . . . 164
9.4.1.65 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_ANT_DIV 164
9.4.1.66 __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_ANT_DIV 164
9.4.1.67 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_ANTENNA . . . 164
9.4.1.68 __QAPI_WLAN_PARAM_GROUP_WIRELESS_DISABLE_CHANNEL 165
9.4.1.69 __QAPI_WLAN_PARAM_GROUP_WIRELESS_EXT_BOARDDATA_INDEX . . . 165
9.4.1.70 __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_STA_DTIM . . . 165
9.4.1.71 __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_WPS_SIMPLE_CONFIGURATION_STATE . . . 165
9.4.1.72 __QAPI_WLAN_PARAM_GROUP_WIRELESS_ROAM_CONTROL . . 166
9.4.1.73 __QAPI_WLAN_PARAM_GROUP_WIRELESS_CONNECT_POLICY . 166
9.4.1.74 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_BMCAST_RX_FILTER . . . 166
9.4.1.75 __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_STA_GET_BSS_MAX_IDLE_PERIOD . . . 167
9.4.1.76 __QAPI_WLAN_PARAM_GROUP_WIRELESS_CONCURRENCY_MODE . . . 167
9.4.1.77 __QAPI_WLAN_PARAM_GROUP_WIRELESS_MGMT_FRAME_FILTER . . . 167

9.4.1.78 __QAPI_WLAN_PARAM_GROUP_WIRELESS_RTS 168
9.4.1.79 __QAPI_WLAN_PARAM_GROUP_WIRELESS_RTS_RATE_2G . . . 168
9.4.1.80 __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCA_PARAM 168
9.4.1.81 __QAPI_WLAN_PARAM_GROUP_WIRELESS_PER_UPPER_THRESHOLD 168
9.4.1.82 __QAPI_WLAN_PARAM_GROUP_WIRELESS_BA_WINDOW 168
9.4.1.83 __QAPI_WLAN_PARAM_GROUP_WIRELESS_SLOT_TIME 169
9.4.1.84 __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCCA_THRESHOLD 169
9.4.1.85 __QAPI_WLAN_PARAM_GROUP_SECURITY_ENCRYPTION_TYPE 169
9.4.1.86 __QAPI_WLAN_PARAM_GROUP_SECURITY_PMK 169
9.4.1.87 __QAPI_WLAN_PARAM_GROUP_SECURITY_PASSPHRASE 170
9.4.1.88 __QAPI_WLAN_PARAM_GROUP_SECURITY_WEP_KEY_INDEX . . 170
9.4.1.89 __QAPI_WLAN_PARAM_GROUP_SECURITY_WEP_KEY_PAIR . . . 170
9.4.1.90 __QAPI_WLAN_PARAM_GROUP_SECURITY_WPS_CREDENTIALS 171
9.4.1.91 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_METHOD . . . 171
9.4.1.92 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_START 171
9.4.1.93 __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_START 171
9.4.1.94 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_IDENTITY . . 172
9.4.1.95 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_USERNAME . 172
9.4.1.96 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PASSWORD . 172
9.4.1.97 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CA_CER . . . 172
9.4.1.98 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CER 172
9.4.1.99 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PRIVATE_KEY 172
9.4.1.100 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_NO_SERVER_AUTH 172
9.4.1.101 __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_END 173
9.4.1.102 __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_MAIN_START . 173
9.4.1.103 __QAPI_WLAN_PARAM_GROUP_SECURITY_DEBUG_LEVEL . . . 173
9.4.1.104 __QAPI_WLAN_PARAM_GROUP_SECURITY_PRIV 173
9.4.1.105 __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_MAIN_END . . 173
9.4.1.106 __QAPI_WLAN_PARAM_GROUP_SECURITY_SUPPL_MAIN_START 173
9.4.1.107 __QAPI_WLAN_PARAM_GROUP_SECURITY_SUPPL_MAIN_END . 173
9.4.1.108 __QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_START 173
9.4.1.109 __QAPI_WLAN_PARAM_GROUP_SECURITY_SET_RSN_CAP . . . 173
9.4.1.110 __QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_END 173
9.4.1.111 __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_START 173
9.4.1.112 __QAPI_WLAN_PARAM_GROUP_SECURITY_FINITE_CYCLIC_GROUP 174
9.4.1.113 __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_PMK_CACHE . 174
9.4.1.114 __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_END 174
9.4.1.115 __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_END 174
9.4.1.116 __QAPI_WLAN_PARAM_GROUP_SECURITY_PMKID 174
9.4.1.117 __QAPI_WLAN_PARAM_GROUP_SECURITY_GET_RSN_CAP . . . 174
10.1 Fatal error 175

10 Fatal Error Manager **175**
10.1.1 Define Documentation 175

10.1.1.1 QAPI_FATAL_ERR . . . 175
10.1.2 Data Structure Documentation . . . 176
10.1.2.1 struct qapi_Err_const_t . . . 176
10.1.3 Function Documentation . . . 176
10.1.3.1 qapi_err_fatal_internal . . . 176
10.1.4 Variable Documentation . . . 177
10.1.4.1 fname . . . 177
10.1.4.2 line . . . 177
11.1 General Purpose Input/Output interface (GPIO) . . . 178

11 GPIO module 178
11.1.1 Data Structure Documentation . . . 178
11.1.1.1 struct qapi_GPIO_Alt_Config_s . . . 178
11.1.1.2 struct qapi_GPIO_Config_s . . . 178
11.1.1.3 struct qapi_GPIO_CB_List_s . . . 179
11.1.2 Typedef Documentation . . . 179
11.1.2.1 qapi_GPIO_Alt_Config_t . . . 179
11.1.2.2 qapi_GPIO_Config_t . . . 179
11.1.2.3 qapi_GPIO_CB_Data_t . . . 179
11.1.2.4 qapi_GPIO_CB_t . . . 180
11.1.2.5 qapi_GPIO_CB_List_t . . . 180
11.1.3 Enumeration Type Documentation . . . 180
11.1.3.1 qapi_GPIO_Id_t . . . 180
11.1.3.2 qapi_GPIO_Direction_t . . . 180
11.1.3.3 qapi_GPIO_Pull_t . . . 180
11.1.3.4 qapi_GPIO_Drive_t . . . 181
11.1.3.5 qapi_GPIO_Value_t . . . 181
11.1.3.6 qapi_GPIO_Trigger_t . . . 181
11.1.4 Function Documentation . . . 181
11.1.4.1 qapi_GPIO_Config . . . 181
11.1.4.2 qapi_GPIO_Set . . . 182
11.1.4.3 qapi_GPIO_Get . . . 182
11.1.4.4 qapi_GPIO_Enable_Interrupt . . . 182
11.1.4.5 qapi_GPIO_Disable_Interrupt . . . 183
11.1.4.6 qapi_GPIO_Mux_Pin_Get . . . 183
12.1 HFC APIs . . . 184

12 HFC APIs 184
12.1.1 Function Documentation . . . 184
12.1.1.1 qapi_hfc_sendto_host_data_pkt . . . 184
12.1.1.2 qapi_hfc_recvfrom_host_data_pkt . . . 184
12.1.1.3 qapi_hfc_sendto_host_config_pkt . . . 185
12.1.1.4 qapi_hfc_recvfrom_host_msg . . . 185
12.1.1.5 qapi_hfc_set_gpio_assert_info . . . 185
13.1 HTTP client . . . 186

13 HTTP Client 186
13.1.1 Define Documentation . . . 186
13.1.1.1 QAPI_NET_HTTPC_CHUNKED_MASK . . . 186
13.1.1.2 QAPI_NET_HTTPC_WITH_HEADER_MASK . . . 186

13.1.2 Data Structure Documentation . . . 186
13.1.2.1 struct qapi_Ssl_Config . . . 186
13.1.2.2 struct qapi_Ssl_Cert . . . 186
13.1.2.3 struct qapi_Net_HTTPc_Response_t . . . 186
13.1.3 Typedef Documentation . . . 187
13.1.3.1 qapi_Ssl_Config_t . . . 187
13.1.3.2 qapi_Ssl_Cert_t . . . 187
13.1.3.3 qapi_HTTPc_CB_t . . . 187
13.1.4 Enumeration Type Documentation . . . 187
13.1.4.1 qapi_Net_HTTPc_Method_e . . . 187
13.1.4.2 qapi_Net_HTTPc_CB_State_e . . . 187
13.1.5 Function Documentation . . . 188
13.1.5.1 qapi_Net_HTTPc_Start . . . 188
13.1.5.2 qapi_Net_HTTPc_Stop . . . 188
13.1.5.3 qapi_Net_HTTPc_New_sess2 . . . 188
13.1.5.4 qapi_Net_HTTPc_New_sess . . . 189
13.1.5.5 qapi_Net_HTTPc_Free_sess . . . 189
13.1.5.6 qapi_Net_HTTPc_Connect . . . 189
13.1.5.7 qapi_Net_HTTPc_Disconnect . . . 190
13.1.5.8 qapi_Net_HTTPc_Request . . . 190
13.1.5.9 qapi_Net_HTTPc_Tunnel_To_HTTPS . . . 190
13.1.5.10 qapi_Net_HTTPc_Set_Body . . . 191
13.1.5.11 qapi_Net_HTTPc_Set_Param . . . 191
13.1.5.12 qapi_Net_HTTPc_Add_Header_Field . . . 191
13.1.5.13 qapi_Net_HTTPc_Clear_Header . . . 192
13.1.5.14 qapi_Net_HTTPc_Configure_SSL . . . 193
13.1.5.15 qapi_Net_HTTPc_Configure_Cert . . . 193
13.1.5.16 qapi_Net_HTTPc_CB_Enable_Adding_Header . . . 193
13.1.5.17 qapi_Net_HTTPc_Send_Data . . . 193
13.1.5.18 qapi_Net_HTTPc_Send_Chunk . . . 194
13.1.5.19 qapi_Net_HTTPc_Rlease_Pre_allocate_buffer . . . 194
14.1 HTTP server . . . 195

14 HTTP Server 195
14.1.1 Function Documentation . . . 195
14.1.1.1 qapi_get_wifi_cfg . . . 195
14.1.1.2 qapi_web_start . . . 195
14.1.1.3 qapi_web_stop . . . 195
15.1 UART . . . 196

15 UART 196
15.1.1 Define Documentation . . . 196
15.1.1.1 QAPI_I2CM_ERROR . . . 196
15.1.1.2 QAPI_UART_ERROR . . . 196
15.1.1.3 QAPI_UART_ERROR_NULL_PTR_ERROR . . . 196
15.1.1.4 QAPI_UART_ERROR_INVALID_PARAM . . . 196
15.1.1.5 QAPI_UART_ERROR_CFG_PARAM . . . 196
15.1.1.6 QAPI_UART_ERROR_BAUDRATE_CFG . . . 196
15.1.1.7 QAPI_UART_ERROR_SEND_BUSY . . . 196
15.1.1.8 QAPI_UART_ERROR_RECV_BUSY . . . 196

15.1.1.9 QAPI_UART_ERROR_TX_ENQUEUE_SEM_SYNC . . . 196
15.1.1.10 QAPI_UART_ERROR_TX_ENQUEUE_FULL . . . 197
15.1.1.11 QAPI_UART_ERROR_TX_DEQUEUE_EMPTY . . . 197
15.1.1.12 QAPI_UART_ERROR_RX_ENQUEUE_FULL . . . 197
15.1.1.13 QAPI_UART_ERROR_RX_DEQUEUE_SEM_SYNC . . . 197
15.1.1.14 QAPI_UART_ERROR_RX_DEQUEUE_EMPTY . . . 197
15.1.1.15 QAPI_UART_ERROR_TRANSFER_TIMEOUT . . . 197
15.1.1.16 QAPI_UART_ERROR_INPUT_FIFO_UNDER_RUN . . . 197
15.1.1.17 QAPI_UART_ERROR_INPUT_FIFO_OVER_RUN . . . 197
15.1.1.18 QAPI_UART_ERROR_OUTPUT_FIFO_UNDER_RUN . . . 197
15.1.1.19 QAPI_UART_ERROR_OUTPUT_FIFO_OVER_RUN . . . 197
15.1.1.20 QAPI_UART_ERROR_TRANSFER_FORCE_TERMINATED . . . 197
15.1.1.21 QAPI_UART_ERROR_BUS_CLK_CFG_FAIL . . . 197
15.1.1.22 QAPI_UART_ERROR_BUS_GPIO_ENABLE_FAIL . . . 198
15.1.1.23 QAPI_UART_ERROR_CANCEL_TRANSFER_FAIL . . . 198
15.1.1.24 QAPI_UART_ERROR_BOOTSTRAP_CFG_FAIL . . . 198
15.1.1.25 QAPI_UART_ERROR_DEVICE_STATE . . . 198
15.1.2 Data Structure Documentation . . . 198
15.1.2.1 struct qapi_UART_Open_Config_t . . . 198
15.1.3 Enumeration Type Documentation . . . 198
15.1.3.1 qapi_UART_Instance_t . . . 198
15.1.3.2 qapi_UART_Bits_Per_Char_e . . . 199
15.1.3.3 qapi_UART_Num_Stop_Bits_e . . . 199
15.1.3.4 qapi_UART_Parity_Mode_e . . . 199
15.1.4 Function Documentation . . . 199
15.1.4.1 qapi_UART_Close . . . 199
15.1.4.2 qapi_UART_Open . . . 200
15.1.4.3 qapi_UART_Receive . . . 200
15.1.4.4 qapi_UART_Transmit . . . 201
15.1.4.5 qapi_UART_Open_With_Rx_Timeout . . . 201
15.1.5 Variable Documentation . . . 202
15.1.5.1 baud_Rate . . . 202
15.1.5.2 parity_Mode . . . 202
15.1.5.3 bits_Per_Char . . . 202
15.1.5.4 num_Stop_Bits . . . 202
15.1.5.5 enable_Loopback . . . 202
16.1 PRNG APIs . . . 203

16 PRNG 203
16.1.1 Define Documentation . . . 203
16.1.1.1 qapi_prng_get . . . 203
16.1.2 Function Documentation . . . 203
16.1.2.1 qapi_prng_get . . . 203
17.1 RRAM APIs . . . 204

17 RRAM 204
17.1.1 Define Documentation . . . 204
17.1.1.1 RRAM_DEVICE_DONE . . . 204
17.1.1.2 RRAM_DEVICE_FAIL . . . 204
17.1.2 Data Structure Documentation . . . 204

17.1.2.1 struct IDAddr . . . 204
17.1.3 Enumeration Type Documentation . . . 204
17.1.3.1 rram_status_t . . . 204
17.1.4 Function Documentation . . . 204
17.1.4.1 qapi_rram_read . . . 204
17.1.4.2 qapi_rram_write . . . 205
18.1 RTC APIs . . . 206

18 RTC 206
18.1.1 Define Documentation . . . 206
18.1.1.1 NUM_DAYS_PER_YEAR . . . 206
18.1.1.2 DIFF_SEC_1900_1970 . . . 206
18.1.2 Data Structure Documentation . . . 206
18.1.2.1 struct qapi_Time_s . . . 206
18.1.2.2 struct ntp_Time_s . . . 206
18.1.2.3 struct time_zone_s . . . 206
18.1.3 Typedef Documentation . . . 207
18.1.3.1 qapi_Time_t . . . 207
18.1.3.2 ntp_Time_t . . . 207
18.1.3.3 time_zone_t . . . 207
18.1.4 Function Documentation . . . 207
18.1.4.1 qapi_Core_RTC_Julian_Get . . . 207
18.1.4.2 qapi_Core_RTC_Julian_Set . . . 207
18.1.4.3 qapi_Core_RTC_NTP_Get . . . 207
18.1.4.4 qapi_Core_RTC_NTP_Set . . . 208
18.1.4.5 qapi_Core_Time_Zone_Get . . . 208
18.1.4.6 qapi_Core_Time_Zone_Set . . . 208
18.1.4.7 qapi_Core_Obtain_Boot_Reason . . . 209

19 Common APIs 210
19.1 Build information . . . 211
19.1.1 Define Documentation . . . 211
19.1.1.1 QAPI_CHIP_VERSION . . . 211
19.1.1.2 QAPI_BUILD_VERSION . . . 211
19.1.1.3 QAPI_CHIP_VERSION_STR . . . 211
19.1.1.4 QAPI_BUILD_VERSION_STR . . . 211
19.1.1.5 QAPI_VERSION_MAJOR . . . 211
19.1.1.6 QAPI_VERSION_MINOR . . . 211
19.1.1.7 QAPI_VERSION_NIT . . . 211
19.1.1.8 __QAPI_VERSION_MAJOR_MASK . . . 211
19.1.1.9 __QAPI_VERSION_MINOR_MASK . . . 211
19.1.1.10 __QAPI_VERSION_NIT_MASK . . . 211
19.1.1.11 __QAPI_VERSION_MAJOR_SHIFT . . . 211
19.1.1.12 __QAPI_VERSION_MINOR_SHIFT . . . 211
19.1.1.13 __QAPI_VERSION_NIT_SHIFT . . . 211
19.1.1.14 __QAPI_ENCODE_VERSION . . . 211
19.1.2 Data Structure Documentation . . . 212
19.1.2.1 struct qapi_FW_Info_t . . . 212
19.1.3 Function Documentation . . . 212
19.1.3.1 qapi_Get_FW_Info . . . 212

19.2 Status information . . . 213
19.2.1 Typedef Documentation . . . 213
19.2.1.1 qapi_Status_t . . . 213
19.3 Error code formats . . . 214
19.3.1 Define Documentation . . . 214
19.3.1.1 __QAPI_ERR_MOD_OFFSET . . . 214
19.3.1.2 __QAPI_ERR_ENCAP_MOD_ID . . . 214
19.3.1.3 __QAPI_ERROR . . . 214
19.4 Module IDs . . . 215
19.4.1 Define Documentation . . . 215
19.4.1.1 QAPI_MOD_BASE . . . 215
19.4.1.2 QAPI_MOD_UART . . . 215
19.4.1.3 QAPI_MOD_I2C . . . 215
19.4.1.4 QAPI_MOD_SPI . . . 215
19.4.1.5 QAPI_MOD_GPIO . . . 215
19.4.1.6 QAPI_MOD_FTC . . . 215
19.4.1.7 QAPI_MOD_M2MDMA . . . 215
19.4.1.8 QAPI_MOD_TMR . . . 215
19.4.1.9 QAPI_MOD_CRYPTO . . . 215
19.4.1.10 QAPI_MOD_LIC . . . 215
19.4.1.11 QAPI_MOD_FWUP . . . 215
19.4.1.12 QAPI_MOD_NVM . . . 215
19.4.1.13 QAPI_MOD_APPI2C . . . 215
19.4.1.14 QAPI_MOD_CONSOLE . . . 215
19.4.1.15 QAPI_MOD_WIFI . . . 216
19.4.1.16 QAPI_MOD_NETWORKING . . . 216
19.4.1.17 QAPI_MOD_FLASH . . . 216
19.4.1.18 QAPI_MOD_HKADC . . . 216
19.5 Status codes . . . 217
19.5.1 Define Documentation . . . 217
19.5.1.1 QAPI_OK . . . 217
19.5.1.2 QAPI_ERROR . . . 217
19.5.1.3 QAPI_ERR_INVALID_PARAM . . . 217
19.5.1.4 QAPI_ERR_NO_MEMORY . . . 217
19.5.1.5 QAPI_ERR_NO_RESOURCE . . . 217
19.5.1.6 QAPI_ERR_BUSY . . . 217
19.5.1.7 QAPI_ERR_NO_ENTRY . . . 217
19.5.1.8 QAPI_ERR_NOT_SUPPORTED . . . 217
19.5.1.9 QAPI_ERR_TIMEOUT . . . 217
19.5.1.10 QAPI_ERR_BOUNDS . . . 217
19.5.1.11 QAPI_ERR_BAD_PAYLOAD . . . 217
19.5.1.12 QAPI_ERR_EXISTS . . . 217
19.6 Types . . . 218

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 implments 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.
in	*status*	QAPI_OK or error code defined in FWUP return status information.

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 initalization. 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 defined in FWUP return status information.

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 defined in FWUP return status information.

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 defined in FWUP return status information.

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 defined in FWUP return status information.

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 defined in FWUP return status information.

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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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_nums*	Image numbers from firmware descriptor.

Returns

On success, QAPI_OK is returned.
On error, error code defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information 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 defined in FWUP return status information is returned.

4.2 FW upgrade definitions

Modules

- FWD boot type information
- FWUP return status information
- Firmware upgrade APIs

FWD Bit Definition

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

- #define QAPI_FW_UPGRADE_FWD_BIT_GOLDEN (0)
- #define QAPI_FW_UPGRADE_FWD_BIT_CURRENT (1)
- #define QAPI_FW_UPGRADE_FWD_BIT_TRIAL (2)

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_FWD_BIT_GOLDEN (0)

4.2.1.4 #define QAPI_FW_UPGRADE_FWD_BIT_CURRENT (1)

4.2.1.5 #define QAPI_FW_UPGRADE_FWD_BIT_TRIAL (2)

4.2.2 Variable Documentation

4.2.2.1 qapi_Fw_Upgrade_Plugin_Init_t qapi_Fw_Upgrade_Plugin_t::fw_Upgrade_Plugin_Init

Callback to initialize a firmware upgrade.

4.2.2.2 qapi_Fw_Upgrade_Plugin_Recv_Data_t qapi_Fw_Upgrade_Plugin_t::fw_Upgrade_Plugin_Recv_Data

Callback to retrieve data.

4.2.2.3 qapi_Fw_Upgrade_Plugin_Abort_t qapi_Fw_Upgrade_Plugin_t::fw_Upgrade_Plugin_Abort

Firmware upgrade plugin abort callback.

4.2.2.4 qapi_Fw_Upgrade_Plugin_Resume_t qapi_Fw_Upgrade_Plugin_t::fw_Upgrade_Plugin_Resume

Firmware upgrade plugin resume callback.

4.2.2.5 qapi_Fw_Upgrade_Plugin_Fin_t qapi_Fw_Upgrade_Plugin_t::fw_Upgrade_-Plugin_Fin

Firmware upgrade plugin finish callback.

4.3 FWD bit information

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.

4.5.1.2 #define QAPI_FW_UPGRADE_ERR_INVALID_PARAM __QAPI_ERROR(QAPI_MOD_FWUP, 2)

Invalid parameter.

4.5.1.3 #define QAPI_FW_UPGRADE_ERR_NOT_INIT __QAPI_ERROR(QAPI_MOD_FWUP, 3)

Firmware upgrade library is not initialized.

4.5.1.4 #define QAPI_FW_UPGRADE_ERR_INCOMPLETE __QAPI_ERROR(QAPI_MOD_FWUP, 4)

Operation is incomplete.

4.5.1.5 #define QAPI_FW_UPGRADE_ERR_SESSION_IN_PROGRESS __QAPI_ERROR(QAPI_MOD_FWUP, 5)

Firmware upgrade session is inprogress.

4.5.1.6 #define QAPI_FW_UPGRADE_ERR_SESSION_NOT_START __QAPI_ERROR(QAPI_MOD_FWUP, 6)

Firmware upgrade session is not started.

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 ready to enter the Suspend state.

4.5.1.8 #define QAPI_FW_UPGRADE_ERR_SESSION_NOT_SUSPEND __QAPI_ERROR(QAPI_MOD_FWUP, 8)

Firmware upgrade session is not in the Suspend state.

4.5.1.9 #define QAPI_FW_UPGRADE_ERR_SESSION_RESUME_NOT_SUPPORT __QAPI_ERROR(QAPI_MOD_FWUP, 9)

Firmware upgrade session resume is not supported by the plugin.

4.5.1.10 #define QAPI_FW_UPGRADE_ERR_SESSION_CANCELLED __QAPI_ERROR(QAPI_MOD_FWUP, 10)

Firmware upgrade session was cancelled.

4.5.1.11 #define QAPI_FW_UPGRADE_ERR_SESSION_SUSPEND __QAPI_ERROR(QAPI_MOD_FWUP, 11)

Firmware upgrade session was suspended.

4.5.1.12 #define QAPI_FW_UPGRADE_ERR_INTERFACE_NAME_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 12)

Interface name is too long.

4.5.1.13 #define QAPI_FW_UPGRADE_ERR_URL_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 13)

URL is too long.

4.5.1.14 #define QAPI_FW_UPGRADE_ERR_FLASH_NOT_SUPPORT_FW_UPGRADE - __QAPI_ERROR(QAPI_MOD_FWUP, 14)

Not supported firmware upgrade.

4.5.1.15 #define QAPI_FW_UPGRADE_ERR_FLASH_INIT_TIMEOUT __QAPI_ERROR(-QAPI_MOD_FWUP, 15)

Flash initialization timeout.

4.5.1.16 #define QAPI_FW_UPGRADE_ERR_FLASH_READ_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 16)

Flash read failure.

4.5.1.17 #define QAPI_FW_UPGRADE_ERR_FLASH_WRITE_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 17)

Flash write failure.

4.5.1.18 #define QAPI_FW_UPGRADE_ERR_FLASH_ERASE_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 18)

Flash erase failure.

4.5.1.19 #define QAPI_FW_UPGRADE_ERR_FLASH_NOT_ENOUGH_SPACE __QAPI_ERROR(QAPI_MOD_FWUP, 19)

Not enough free space in flash.

4.5.1.20 #define QAPI_FW_UPGRADE_ERR_FLASH_CREATE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 20)

Partition creation failure.

4.5.1.21 #define QAPI_FW_UPGRADE_ERR_FLASH_IMAGE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 21)

Partition image was not found.

4.5.1.22 #define QAPI_FW_UPGRADE_ERR_FLASH_ERASE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 22)

Partition erase failure.

4.5.1.23 #define QAPI_FW_UPGRADE_ERR_FLASH_WRITE_PARTITION __QAPI_ERROR(QAPI_MOD_FWUP, 23)

Partition write failure.

4.5.1.24 #define QAPI_FW_UPGRADE_ERR_GET_PARTITION_NULL __QAPI_ERROR(QAPI_MOD_FWUP, 24)

NULL partition.

4.5.1.25 #define QAPI_FW_UPGRADE_ERR_REACH_MAX_IMAGE_ENTRY __QAPI_ERROR(QAPI_MOD_FWUP, 25)

Reach max image entry.

4.5.1.26 #define QAPI_FW_UPGRADE_ERR_IMAGE_UNUSED __QAPI_ERROR(QAPI_MOD_FWUP, 26)

Image Entry is not used

4.5.1.27 #define QAPI_FW_UPGRADE_ERR_IMAGE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 27)

Image not found failure.

4.5.1.28 #define QAPI_FW_UPGRADE_ERR_IMAGE_DOWNLOAD_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 28)

Image download failure.

4.5.1.29 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_CHECKSUM __QAPI_ERROR(QAPI_MOD_FWUP, 29)

Incorrect image checksum failure.

4.5.1.30 #define QAPI_FW_UPGRADE_ERR_SERVER_RSP_TIMEOUT __QAPI_ERROR(QAPI_MOD_FWUP, 30)

Server communication timeout.

4.5.1.31 #define QAPI_FW_UPGRADE_ERR_INVALID_FILENAME __QAPI_ERROR(QAPI_MOD_FWUP, 31)

Image file name is invalid.

4.5.1.32 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_HDR __QAPI_ERROR(QAPI_MOD_FWUP, 32)

Firmware upgrade image header is invalid.

4.5.1.33 #define QAPI_FW_UPGRADE_ERR_INSUFFICIENT_MEMORY __QAPI_ERROR(QAPI_MOD_FWUP, 33)

Not enough memory.

4.5.1.34 #define QAPI_FW_UPGRADE_ERR_INCORRECT_SIGNATURE __QAPI_ERROR(QAPI_MOD_FWUP, 34)

Firmware upgrade image signature is invalid.

4.5.1.35 #define QAPI_FW_UPGRADE_ERR_INCORRCT_VERSION __QAPI_ERROR(QAPI_MOD_FWUP, 35)

Firmware upgrade image version is invalid.

4.5.1.36 #define QAPI_FW_UPGRADE_ERR_INCORRECT_NUM_IMAGES __QAPI_ERROR(QAPI_MOD_FWUP, 36)

Firmware upgrade image number of images is invalid.

4.5.1.37 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_LENGTH __QAPI_ERROR(QAPI_MOD_FWUP, 37)

Firmware upgrade image length is invalid.

4.5.1.38 #define QAPI_FW_UPGRADE_ERR_INCORRECT_HASH_TYPE __QAPI_ERROR(QAPI_MOD_FWUP, 38)

Firmware upgrade image hash type is invalid.

4.5.1.39 #define QAPI_FW_UPGRADE_ERR_INCORRECT_IMAGE_ID __QAPI_ERROR(QAPI_MOD_FWUP, 39)

Firmware upgrade image ID is invalid.

4.5.1.40 #define QAPI_FW_UPGRADE_ERR_SBL_ONLY_NOT_SUPPORT __QAPI_ERROR(QAPI_MOD_FWUP, 40)

SBL upgrade only is not supported.

4.5.1.41 #define QAPI_FW_UPGRADE_ERR_SBL_NOT_SUPPORT_UPGRADE __QAPI_ERROR(QAPI_MOD_FWUP, 41)

SBL upgrade not supported.

4.5.1.42 #define QAPI_FW_UPGRADE_ERR_SBL_NOT_ENOUGH_SPACE __QAPI_ERROR(QAPI_MOD_FWUP, 42)

No enough memory for SBL upgrade.

4.5.1.43 #define QAPI_FW_UPGRADE_ERR_INVALID_FDT __QAPI_ERROR(QAPI_MOD_FWUP, 43)

Invalid FDT.

4.5.1.44 #define QAPI_FW_UPGRADE_ERR_BATTERY_LEVEL_TOO_LOW __QAPI_ERROR(QAPI_MOD_FWUP, 44)

Battery level is too low.

4.5.1.45 #define QAPI_FW_UPGRADE_ERR_CRYPTO_FAIL __QAPI_ERROR(QAPI_MOD_FWUP, 45)

Crypto check failure.

4.5.1.46 #define QAPI_FW_UPGRADE_ERR_PLUGIN_ENTRY_EMPTY __QAPI_ERROR(QAPI_MOD_FWUP, 46)

Firmware upgrade plugin callback is empty.

4.5.1.47 #define QAPI_FW_UPGRADE_ERR_TRIAL_IS_RUNNING __QAPI_ERROR(QAPI_MOD_FWUP, 47)

Trial image is running

4.5.1.48 #define QAPI_FW_UPGRADE_ERR_FILE_NOT_FOUND __QAPI_ERROR(QAPI_MOD_FWUP, 48)

File was not found.

4.5.1.49 #define QAPI_FW_UPGRADE_ERR_FILE_OPEN_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 49)

Open file failure.

4.5.1.50 #define QAPI_FW_UPGRADE_ERR_FILE_NAME_TOO_LONG __QAPI_ERROR(QAPI_MOD_FWUP, 50)

File name is too long.

4.5.1.51 #define QAPI_FW_UPGRADE_ERR_FILE_WRITE_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 51)

Write file failure.

4.5.1.52 #define QAPI_FW_UPGRADE_ERR_MOUNT_FILE_SYSTEM_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 52)

Mount file system failure.

4.5.1.53 #define QAPI_FW_UPGRADE_ERR_CREATE_THREAD_ERROR __QAPI_ERROR(QAPI_MOD_FWUP, 53)

Firmware upgrade create thread failure.

4.5.1.54 #define QAPI_FW_UPGRADE_ERR_PRESERVE_LAST_FAILED __QAPI_ERROR(QAPI_MOD_FWUP, 54)

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	*flash_info*	Pointer to flash device info data.

Returns

QAPI_OK – On success.
Error code – On failure.

5.2 Flash config APIs

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_enter_sleep (uint8_t *enable,* uint32_t *wait_time,* uint32_t *sleep_time*)

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

Parameters

in	*enable*	Other parameters are valid only when enable is 1; **Supported values:** • 1 – Enable • 0 – Disable
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 starts. If a successful connection happens, cancels the timer. If timeout, the system determines whether to enter deepsleep.
in	*wmi_wait*	wmi_wait time in ms. Upon recnx_wait timeout, checks if any WMI commands were received during the wmi_wait duration. If no, sends the system to deepsleep. If yes, starts a timer with wmi_wait duration.
in	*cnx_wait*	Time in ms. Use for ENABLE_IMPS_TIMER_ON_BOOTUP feature, which means starting this timer during bootup, if there's no wlan connection during this period, then the system enters deepsleep.

Returns

- QAPI_OK – IMPS configured and enabled successfully.

8.1.2.4 qapi_Status_t qapi_imps_disable_sleep (void)

Disables IMPS.

Returns

- QAPI_OK – IMPS Disenable successfully.

8.1.2.5 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.

8.1.2.6 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.

8.1.2.7 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.

8.1.2.8 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.

8.1.2.9 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 Define Documentation

9.1.1.1 #define __QAPI_WLAN_DEFAULT_SCAN_CTRL_FLAGS (QAPI_WLAN_CONNECT_SCAN_CTRL_FLAGS_E| QAPI_WLAN_SCAN_CONNECTED_CTRL_FLAGS_E| QAPI_WLAN_ACTIVE_SCAN_CTRL_FLAGS_E| QAPI_WLAN_ROAM_SCAN_CTRL_FLAGS_E| QAPI_WLAN_ENABLE_AUTO_CTRL_FLAGS_E)

Macro to use the default scan control flags for a scan.

9.1.1.2 #define __QAPI_WLAN_PMKID_LEN 16

Size of the WLAN PMKID in bytes.

9.1.1.3 #define __QAPI_WLAN_DBGLOG_MISC_CONFIG_FLUSH_FLAG_BIT_OFFSET 0

As part of misc_Config, start bit offset of flush, to be used to flush the dbglogs in current buffer to the host.

9.1.1.4 #define __QAPI_WLAN_DBGLOG_MISC_CONFIG_FLUSH_FLAG_MASK 0x01

As part of misc_Config, range mask of flush, to be used to flush the dbglogs in current buffer to the host.

9.1.1.5 #define QAPI_TX_POWER_RESTORE 100

Macro definitions for restoring TX power to default value.

9.1.1.6 #define MIN_GEN_PKT_INTERVAL 100

Macro definitions for packet generation.

9.1.2 Data Structure Documentation

9.1.2.1 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.

Type	Parameter	Description
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.

9.1.2.2 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.
uint8_t	bssid[__QAPI-_WLAN_MA-C_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.

9.1.2.3 struct qapi_WLAN_BSS_Scan_Info_ExtV1_t

qapi_WLAN_BSS_Scan_Info_t is not enough and cannot be changed, this structure reports more information.

Data fields

Type	Parameter	Description
qapi_WLAN-_BSS_Scan_-Info_t	info	The basic info.
int32_t	is_beacon	TRUE: beacon, FALSE: ProbeRsp.
const uint8_t *	beacon_IEs	Beacon or Probe Response IEs which do not include fixed fields.
uint16_t	beacon_IEs_-Len	Length of beacon_IEs.
uint8_t	reserved[6]	Reserved, 2 bytes for allignment, 4 bytes for extension.

9.1.2.4 struct qapi_WLAN_Evt_Hdr_t

Overall status.

Data fields

Type	Parameter	Description
qapi_Status_t	status	Status.

9.1.2.5 struct qapi_WLAN_Enable_Evt_t

WLAN enable event.

Data fields

Type	Parameter	Description
qapi_WLAN_-Evt_Hdr_t	evt_hdr	Contains the common event header.
uint8_t	mac_addr[__Q-API_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.

9.1.2.6 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.

9.1.2.7 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.

9.1.2.8 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.

9.1.2.9 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

9.1.2.10 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	
uint16_t	reserved2	Channel frequency. Reserved.

9.1.2.11 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.

9.1.2.12 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 .

9.1.2.13 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)
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

Type	Parameter	Description
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,

9.1.2.14 struct qapi_WLAN_Connect_Cb_Info_ExtV1_t

qapi_WLAN_Connect_Cb_Info_t is not enough and cannot be changed, this structure reports more information.

Data fields

Type	Parameter	Description
qapi_WLAN_-Connect_Cb_-Info_t	info	The basic info.
const uint8_t *	req_IEs	(Re)Association Request IEs which do not include fixed fields.
const uint8_t *	resp_IEs	(Re)Association Response IEs which do not include fixed fields.
const uint8_t *	beacon_IEs	Now only include WPA or RSN IE.
uint8_t	req_IEs_Len	Length of req_IEs.
uint8_t	resp_IEs_Len	Length of resp_IEs.
uint8_t	beacon_IEs_-Len	Length of beacon_IEs.
uint8_t	channel	Wireless channel.
uint16_t	listen_Interval	listen interval.
uint16_t	beacon_Interval	beacon interval.
uint8_t	reserved[8]	Reserved, 2 bytes for internal member, 2 bytes for allignment, 4 bytes for extension.

9.1.2.15 struct qapi_WLAN_Ready_Cb_Info_t

Data structure used to indicate the WLAN enablement status to a coexistence subsystem. This callback is not passed on for the application, but only for other internal subsystems.

Data fields

Type	Parameter	Description
uint32_t	numDevices	Supported number of virtual interfaces/devices.

9.1.2.16 struct qapi_WLAN_Device_Stats_t

Data structure that represents WLAN statistics information for each virtual device. This data structure contains counts of various types of packets transmitted/received on the requested virtual device.

Data fields

Type	Parameter	Description
uint32_t	unicast_Tx_-Pkts	Unicast Tx packets. Only valid in STA mode.
uint32_t	unicast_Rx_-Pkts	Unicast Rx packets. Only valid in STA mode.
uint32_t	multicast_Tx_-Pkts	Multicast Tx packets. Only valid in STA mode.
uint32_t	multicast_Rx_-Pkts	Multicast Rx packets. Only valid in STA mode.
uint32_t	broadcast_Tx_-Pkts	Broadcast Tx packets. Only valid in STA mode.
uint32_t	broadcast_Rx_-Pkts	Broadcast Rx packets. Only valid in STA mode.
uint32_t	unicast_Non_-Null_Tx_Pkts	Unicast Tx packets excluding NULL and QoS NULL packets. Only valid in STA mode.
uint32_t	unicast_Non_-Null_Rx_Pkts	Unicast Rx packets excluding NULL and QoS NULL packets. Only valid in STA mode.
uint32_t	unicast_-Filtered_-Accepted_Tx_-Pkts	Unicast filtered and accepted Tx packets. Only valid in STA mode. Must enable WoW.
uint32_t	unicast_-Filtered_-Accepted_Rx_-Pkts	Unicast filtered and accepted Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	multicast_-Filtered_-Accepted_Tx_-Pkts	Multicast filtered and accepted Tx packets. Only valid in STA mode. Must enable WoW.
uint32_t	multicast_-Filtered_-Accepted_Rx_-Pkts	Multicast filtered and accepted Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	broadcast_-Filtered_-Accepted_Tx_-Pkts	Broadcast filtered and accepted Tx packets. Only valid in STA mode. Must enable WoW.
uint32_t	broadcast_-Filtered_-Accepted_Rx_-Pkts	Broadcast filtered and accepted Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	unicast_-Filtered_-Rejected_Tx_-Pkts	Unicast filtered and rejected Tx packets. Only valid in STA mode. Must enable WoW.

Type	Parameter	Description
uint32_t	unicast_-Filtered_-Rejected_Rx_-Pkts	Unicast filtered and rejected Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	multicast_-Filtered_-Rejected_Tx_-Pkts	Multicast filtered and rejected Tx packets. Only valid in STA mode. Must enable WoW.
uint32_t	multicast_-Filtered_-Rejected_Rx_-Pkts	Multicast filtered and rejected Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	broadcast_-Filtered_-Rejected_Tx_-Pkts	Broadcast filtered and rejected Tx packets. Only valid in STA mode. Must enable WoW.
uint32_t	broadcast_-Filtered_-Rejected_Rx_-Pkts	Broadcast filtered and rejected Rx packets. Only valid in STA mode. Must enable WoW.
uint32_t	null_Tx_Pkts	NULL Tx packets. Valid in AP and STA modes.
uint32_t	null_Rx_Pkts	NULL Rx packets. Valid in AP and STA modes.
uint32_t	qos_Null_Tx_-Pkts	QOS NULL Tx packets. Valid in AP and STA modes.
uint32_t	qos_Null_Rx_-Pkts	QOS NULL Rx packets. Valid in AP and STA modes.
uint32_t	ps_Poll_Tx_-Pkts	PS Poll Tx packets. Valid in STA mode.
uint32_t	ps_Poll_Rx_-Pkts	PS Poll Rx packets. Valid in AP and STA modes.
uint32_t	tx_Retry_Cnt	Tx retry count. Only valid in STA mode.
uint32_t	beacon_Miss_-Cnt	Beacon miss count. Only valid in STA mode.
uint32_t	beacons_-Received_Cnt	Received beacon miss count. Only valid in STA mode.
uint32_t	beacon_-Resync_-Success_Cnt	Beacon resync success count. Only valid in STA mode.
uint32_t	beacon_-Resync_-Failure_Cnt	Beacon resync failure count. Only valid in STA mode.
uint32_t	curr_Early_-Wakeup_Adj_-In_Ms	Current early wakeup adjustment. Only valid in STA mode.
uint32_t	avg_Early_-Wakeup_Adj_-In_Ms	Average early wakeup adjustment. Only valid in STA mode.

Type	Parameter	Description
uint32_t	early_-Termination_-Cnt	Early termination count. Only valid in STA mode.
uint32_t	uapsd_Trigger-_Rx_Cnt	UAPSD trigger Rx count. Only valid in STA mode.
uint32_t	uapsd_Trigger-_Tx_Cnt	UAPSD trigger Tx count. Only valid in STA mode.

9.1.2.17 struct qapi_WLAN_Common_TxRx_Buffer_Info

Data structure that represents buffer inforamtion for WLAN Transmit/Receive. The buffer information in this data structure are independent of the virtual devices.

Data fields

Type	Parameter	Description
uint8_t	htc_inf_cur_cnt	The count of current available buffers in htc interface.
uint8_t	htc_inf_reaped-_cnt	The count of reaped buffers in htc interface.
uint8_t	mac_inf_cur_-cnt	The count of current available buffers in mac interface.
uint8_t	mac_inf_-reaped_cnt	The count of reaped buffers in mac interface.
uint8_t	fw_inf_cur_cnt	The count of current available buffers in fw interface.
uint8_t	fw_inf_reaped-_cnt	The count of reaped buffers in fw interface.
uint8_t	free_buf_cnt	The count of free buffers in buffer pool.
uint8_t	mgmt_buf_cnt	The count of remaining management buffers.
uint8_t	smmgmt_buf_-cnt	The count of remaining small management buffers.
uint8_t	num_txbuf_-queued	The number of buffers queued for tx.
uint8_t	num_rxbuf_-queued	The number of buffers queued for rx.
uint8_t	reserved	

9.1.2.18 struct qapi_WLAN_Common_Stats_t

Data structure that represents system level statistics information of the WLAN subsystem. The statistics information in this data structure are independent of the virtual devices.

Data fields

Type	Parameter	Description
uint32_t	total_Active_-Time_In_Ms	Total time in milliseconds for which the WLAN subsystem has been Active.
uint32_t	total_-Powersave_-Time_In_Ms	Total time in milliseconds for which the WLAN subsystem has been in Power Save.

Type	Parameter	Description
qapi_WLAN_-Common_Tx-Rx_Buffer_Info	txrx_buffer_-info	The buffer information for WLAN transmit/receive.
uint32_t	FCS_Error_Rx-_Pkts	Receive FCE error packets.

9.1.2.19 struct qapi_WLAN_Device_Stats_Ext_t

Data structure that represents extensional WLAN statistics information for each virtual device. This data structure contains the number of amsdu packets received on the requested virtual device.

Data fields

Type	Parameter	Description
uint32_t	rx_amsdu_pkts	
uint32_t	reserved	Number of received AMSDU packets. Valid in AP and STA modes.

9.1.2.20 struct qapi_WLAN_Device_Stats_Ext2_t

Data structure that represents extensional WLAN statistics information for each virtual device on version 2.

Data fields

Type	Parameter	Description
uint16_t	wmi_event_-missed_last	The last missed WMI event. Valid in AP and STA modes.
uint16_t	reserved	Reserved.
uint32_t	wmi_event_-missed_bitmap	Bitmap of missed WMI events. Valid in AP and STA modes.
uint32_t	wmi_event_-missed_cnt	Count of total missed WMI events. Valid in AP and STA mode.
uint32_t	valid_Rx_Pkts	Receive valid packets. Only valid in STA mode.
uint32_t	addr_Miss_-Match_Rx_Pkts	Receive valid packets but filtered out by one of the filters. Valid in AP and STA modes.
uint32_t	avarage_Rssi_-Data_Pkts	Average RSSI for all valid data packets received. Valid in AP and STA modes.
uint32_t	avarage_Rssi_-Mgmt_Pkts	Average RSSI for all valid management packets received. Valid in AP and STA modes.

9.1.2.21 struct qapi_WLAN_Statistics_t

Data structure that represents system and MAC statistics of the WLAN subsytem. This structure includes both virtual device-specific statistics and virtual device independent statistics.

Data fields

Type	Parameter	Description
qapi_WLAN_-Device_Stats_t	dev_Stats	Per device statistics.

Type	Parameter	Description
qapi_WLAN-_Common_-Stats_t	common_Stats	Statistics common to all devices.
qapi_WLAN_-Device_Stats_-Ext_t	dev_Stats_Ext	Per device extensional statistics.
qapi_WLAN_-Device_Stats_-Ext2_t	dev_Stats_Ext2	Per device extensional statistics for version 2.

9.1.2.22 struct qapi_WLAN_Get_Statistics_t

Data structure used to retrieve WLAN statistics from the driver. The application passes da ata structure of this type to get the statistics information by calling qapi_WLAN_Get_Param() with the command ID as __QAPI_WLAN_PARAM_GROUP_WIRELESS_STATS.

Data fields

Type	Parameter	Description
uint8_t	reset_Counters-_Flag	[IN] 1: Reset statistics counters after getting the current count; 0: Do not reset counters
qapi_WLAN_-Statistics_t *	wlan_Stats_-Data	[OUT] Data structure that is to be filled by the driver that actually holds the statistics information.

9.1.2.23 struct qapi_WLAN_Get_Channel_List_t

Data structure used to retrieve the current regulatory domain channel list. The application passes a data structure of this type to get the channel list information by calling qapi_WLAN_Get_Param() with the command ID as __QAPI_WLAN_MAX_NUM_CUR_REGDOAMIN_CHANLIST_CHNNELS.

When using the getRegDomainChannelList() command, this data structure should be allocated from heap memory (not from the stack).

Data fields

Type	Parameter	Description
uint16_t	channel_List[__QAPI_WLA-N_MAX_NU-M_CUR_REG-DOAMIN_CH-ANLIST_CH-NNELS]	WLAN Channel list array
uint8_t	number_Of_-Channels	Number of channels supported in the current regulatory setting.

9.1.2.24 struct qapi_WLAN_Preferred_Network_Offload_Info_t

Data structure that the application uses to interpret the event payload information on receiving QAPI_WLAN_PREFERRED_NETWORK_OFFLOAD_CB_E from the WLAN driver. The event data provides the result of the PNO operation.

Data fields

Type	Parameter	Description
uint8_t	profile_-Matched	A value of 1 indicates that a matching profile has been found on the PNO scan. A value of 0 indicates that a matching profile has not been found on the PNO scan.
uint8_t	matched_Index	Index of a matched profile. This index number is the same number given by the application when adding an AP profile to the PNO profile list.
int32_t	rssi	RSSI of the AP found that corresponds to the matching profile index.
uint32_t	num_Fast_-Scans_-Remaining	Number of fast scans remaining.

9.1.2.25 struct qapi_WLAN_Event_Filter_t

Data structure that enables applications to filter unnecessary events from waking up the application processor.

The applications can program event filters by invoking qapi_WLAN_Set_Param() with command ID __QAPI_WLAN_PARAM_GROUP_WIRELESS_EVENT_FILTER.

This data structure does not allow applications to incrementally add filter events. It is only possible to enable or disable filtering for all events at once.

Data fields

Type	Parameter	Description
qapi_WLAN_-Enable_e	action	1 to start filtering a list of events provided, 0 to disable filtering.
uint32_t	num_Events	Number of events provided by the application in event[].
qapi_WLAN-_Filterable_-Event_e	event[__QAPI-_WLAN_MA-X_NUM_FIL-TERED_EVE-NTS]	Actual list of events to be filtered. The events being provided should be of type qapi_WLAN_Filterable_Event_e and the number of events in this list should not exceed __QAPI_WLAN_MAX_NUM_FILTERED_EVENTS.

9.1.2.26 struct qapi_WLAN_Firmware_Version_String_t

Data structure to get the WLAN host/firmware version information in string format.

Data fields

Type	Parameter	Description
uint8_t	host_Version[__QAPI_WLAN_VERSION_SUBSTRING_LEN]	Not used, Host version.
uint8_t	target_Version[__QAPI_WLAN_VERSION_SUBSTRING_LEN]	Not used, Target version.
uint8_t	wlan_Version[__QAPI_WLAN_VERSION_SUBSTRING_LEN]	WLAN version.
uint8_t	abi_Version[__QAPI_WLAN_VERSION_SUBSTRING_LEN]	Device ABI version.

9.1.2.27 struct qapi_WLAN_Scan_Params_t

Data structure that enables the application to configure scan parameters for subsequent scan operations.

Data fields

Type	Parameter	Description
uint16_t	fg_Start_Period	Foreground scan start period in seconds. If this parameter is set to 0, the period is set to the default value of 1 second. If this parameter is set to 0xFFFF, the periodic background scan is disabled.
uint16_t	fg_End_Period	Foreground scan end period in seconds. If this parameter is set to 0, the period is set to the default value of 60 seconds.
uint16_t	bg_Period	Background scan period in seconds. Stations can use this mode to scan for neighboring APs without disconnecting from the AP to which they are currently connected. If this parameter is set to 0, the period is set to the default value of 60 seconds. If this parameter is set to 0xFFFF, the periodic background scan is disabled.
uint16_t	max_Act_Chan_Dwell_Time_In_Ms	Active channel maximum dwell time in ms. If this parameter is set to 0, the default period is 20 ms.

Type	Parameter	Description
uint16_t	passive_Chan_-Dwell_Time_-In_Ms	Channel dwell time for passive scan mode in which the client waits at each channel to receive beacon frames from APs within the proximity. If set to 0, the period is set to the default value of 50 ms.
uint8_t	short_Scan_-Ratio	Ratio of the number of short scans per long scan for a foreground search. The default value is 3. Short scan is done only in the channels with configured SSIDs, whereas long scan is done in all channels.
uint8_t	scan_Ctrl_-Flags	Scan control flags. • 0x00 – Use the current scan control settings. • 0xFF – Disable all scan control flags; this will set the flag to 0. For setting specific scan control flags, use each bit in this byte-sized field as one flag. For each bit, 1 = TRUE and 0 = FALSE. • Bit0 – Scan is allowed during connection. • Bit1 – Scan is allowed for the SSID station if it is already connected. • Bit2 – Enable an active scan. • Bit3 – Scan is allowed for roaming when a beacon misses or a low signal strength is identified. • Bit4 – Follow the customer BSSINFO reporting rule. • Bit5 – Device is to scan after a disconnection. • Bit6 – To be implemented, Scan complete event with a cancelled status is generated when a scan is preempted before it is completed. • Bit7 – Scanning DFS channels is to be skipped.
uint16_t	min_Act_Chan-_Dwell_Time_-In_Ms	Minimum dwell time at each channel in case there is no activity during Active Scan mode. The default value is 0, where each channel follows the period defined by the maximum active channel dwell time.
uint16_t	max_Act_Scan-_Per_Ssid	Maximum number of scans allowed per channel to search for configured SSIDs during Active mode. Default value is 1.
uint32_t	max_Dfs_-Chan_Act_-Time_In_Ms	Maximum allowed time for scanning in DFS channels. The default is 2000 ms.

9.1.2.28 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

Type	Parameter	Description
qapi_WLAN_-Power_Module-_e	power_Module	Module requesting power mode change

9.1.2.29 struct qapi_WLAN_Power_Policy_Params_t

Data structure to configure WLAN power policy information.

See also

qapi_WLAN_Set_Param
__QAPI_WLAN_PARAM_GROUP_WIRELESS_POWER_MODE_POLICY

Data fields

Type	Parameter	Description
uint16_t	idle_Period_In-_Ms	Idle period during traffic after which WLAN can go to sleep. The system default for this parameter is 200 ms.
uint16_t	ps_Poll_Num	This parameter dictates the number of contiguous PS-Poll frames that the WLAN firmware is to send before sending an 802.11 NULL frame to indicate a power save exit to the access point. The system default for this parameter is 1.
qapi_WLAN_-DTIM_Policy_-e	dtim_Policy	DTIM policy to be used. The application can select one of the DTIM policies from qapi_WLAN_DTIM_Policy_e. The system default for this parameter is QAPI_WLAN_DTIM_NORMAL_E.
qapi_WLAN_-TX_Wakeup_-Policy_e	tx_Wakeup_-Policy	Tx wake-up policy to be used. The application can select one of the Tx wake-up policies from qapi_WLAN_TX_Wakeup_Policy_e. The system default for this parameter is QAPI_WLAN_TX_WAKEUP_ON_SLEEP_E.
uint16_t	num_Tx_To_-Wakeup	Number of contiguous transmit packets after which the WLAN firmware is to exit Protocol Power Save mode. This parameter is effective only if tx_Wakeup_Policy is set to QAPI_WLAN_TX_W-AKEUP_ON_SLEEP_E. The system default for this parameter is 1.
qapi_WLAN_-Power_Save_-Policy_e	ps_Fail_Event-_Policy	Policy to indicate a power save entry/exit failure event. The application can select one of the policies from qapi_WLAN_Power_Save_Policy_e. The system default for this parameter is QAPI_WLAN_PS_SEND_POWER_SAVE_FAIL_E-VENT_ALWAYS_E.

9.1.2.30 struct qapi_WLAN_Rssi_Threshold_Params_t

Data structure for RSSI thresholds.

Data fields

Type	Parameter	Description
uint32_t	poll_Time	Polling time as a factor of LI.
int16_t	threshold_-Above1_Val	Lowest of the upper thresholds.

Type	Parameter	Description
int16_t	threshold_-Above2_Val	Higher than above1.
int16_t	threshold_-Above3_Val	Higher than above2.
int16_t	threshold_-Above4_Val	Higher than above3.
int16_t	threshold_-Above5_Val	Higher than above5.
int16_t	threshold_-Above6_Val	Highest of the upper thresholds.
int16_t	threshold_-Below1_Val	Lowest of lower thresholds.
int16_t	threshold_-Below2_Val	Higher than below1.
int16_t	threshold_-Below3_Val	Higher than below2.
int16_t	threshold_-Below4_Val	Higher than below3.
int16_t	threshold_-Below5_Val	Higher than below4.
int16_t	threshold_-Below6_Val	Highest of the lower thresholds.
uint8_t	weight	Alpha.

9.1.2.31 struct qapi_WLAN_LowRssi_RoamControl_Params_t

Data structure for low RSSI control.

Data fields

Type	Parameter	Description
int16_t	lowrssi_scan_-threshold	Roam low RSSI threshold.
uint16_t	reserved[2]	Reserved.

9.1.2.32 struct qapi_WLAN_Channel_Switch_t

Data structure to be used if the application wants to perform a channel switch after a given number of beacon intervals.

A channel switch request can be issued only when the device is operating in SoftAP mode.

On receiving this request, the WLAN firmware starts adding channel switch information elements in its beacons with the necessary information until the channel switch operation is performed.

Data fields

Type	Parameter	Description
uint32_t	channel	New wireless channel frequency to which the SoftAP should switch.

Type	Parameter	Description
uint8_t	tbtt_Count	Number of beacon intervals after which the channel switch is to be performed.

9.1.2.33 struct qapi_WLAN_TCP_Offload_Enable_t

Data structure that enables the application to enable/disable the TCP Keepalive offload feature along with the ability to configure session-independent parameters of the TCP Keepalive offload feature.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_OFFLOAD_ENABLE
qapi_WLAN_Set_Param

Data fields

Type	Parameter	Description
uint8_t	enable	Used to enable (1) or disable (0) the TCP Keepalive offload feature.
uint16_t	keepalive_-Interval	TCP Keepalive interval in seconds. The keepalive frames for every configured TCP session are sent periodically with this interval as periodicity.
uint16_t	keepalive_Ack-_Recv_-Threshold	Number of keepalive frames to try before confirming that the remote connection is dead if TCP acknowledgements are not received from the peer.

9.1.2.34 struct qapi_WLAN_TCP_Offload_Config_Params_t

Data structure to configure parameters for a TCP session for which the keepalive transmission is to be offloaded to the firmware.

The application should ensure that maximum number of offloaded TCP KA sessions does not exceed 3.

This data structure allows parameters to be configured for one TCP session by invoking qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_OFFLOAD_SESSION_CFG as the command ID.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_OFFLOAD_SESSION_CFG
qapi_WLAN_Set_Param

Data fields

Type	Parameter	Description
int32_t	sock_Id	Socket handle of the TCP session to be offloaded.
int16_t	src_Port	Source port of the TCP session in host byte order.
int16_t	dst_Port	Destination port of the TCP session in host byte order.
uint32_t	src_IP	Source IPv4 address of the TCP session in host byte order if ip_Protocol_Type is of type IPv4.
uint32_t	dst_IP	Destination IPv4 address of the TCP session in host byte order if ip_Protocol_Type is of type IPv4.

Type	Parameter	Description
uint8_t	dest_MAC[__QAPI_WLAN_MAC_LEN]	MAC address of the TCP peer that corresponds to the destination IP address.
int32_t	sequence_Num	TCP sequence number of the TCP session at the time of enabling the KA offload.
int32_t	ack_Sequence_Num	TCP acknowledgement number of the TCP session at the time of enabling the KA offload.
qapi_WLAN_IP_Protocol_Type_e	ip_Protocol_Type	IP protocol type of the socket: IPv4/IPv6 (should be one of the types in qapi_WLAN_IP_Protocol_Type_e).
uint8_t	src_IP_v6addr[__QAPI_WLAN_IPV6_ADDR_LEN]	Source IPv6 address of the TCP session if ip_Protocol_Type is of type IPv6.
uint8_t	dst_IP_v6addr[__QAPI_WLAN_IPV6_ADDR_LEN]	Destination IPv6 address of the TCP session if ip_Protocol_Type is of type IPv6.

9.1.2.35 struct qapi_WLAN_TCP_Keepalive_Event_Info_t

Data structure to store TCP keepalive event information.

This data structure represents the session-specific information when a TCP KA offload event is received from the WLAN firmware. The combination of socket ID, source port, and destination port can be used to uniquely identify a TCP session.

Data fields

Type	Parameter	Description
int32_t	sock_Id	Socket ID of the TCP session for which the event is received.
uint32_t	sequence_Num	Updated TCP sequence number after transmitting a few TCP KA frames from the firmware.
uint32_t	ack_Sequence_Num	Updated TCP acknowledgement number after transmitting a few TCP KA frames from the firmware.
uint16_t	src_Port	Source port of the TCP session for which the event is received.
uint16_t	dst_Port	Destination port of the TCP session for which the event is received.
uint8_t	status	Status of the TCP KA offload for this TCP session: • A value of 1 indicates that the peer did not transmit an acknowledgment for the keepalive frames transmitted by the firmware • A value of 2 indicates that the TCP KA offload feature has been disabled by the application. An event with this status is needed to ensure that the TCP sequence number and the TCP acknowledgement number are synchronized between the TCP stack in the host and the offloaded TCP KA offload feature in the WLAN firmware.

9.1.2.36 struct qapi_WLAN_TCP_Keepalive_Event_t

Data structure to interpret the event payload when the WLAN driver invokes an application-registered callback with the event ID QAPI_WLAN_TCP_KEEPALIVE_OFFLOAD_CB_E.

This event is received in two cases from the firmware – when TCP ACK is not received from the peer for the keepalive frames sent by the firmware, and when the TCP KA feature is disabled by the application.

Data fields

Type	Parameter	Description
uint32_t	session_cnt	Number of TCP KA sessions for which information is available in the event_info parameter.
qapi_WLAN_TCP_Keepalive_Event_Info_t	event_info[1]	Flexible array that holds the TCP KA event information for one or more TCP sessions. The number of sessions is available in the session_cnt parameter.

9.1.2.37 struct qapi_WLAN_A_Netbuf_Pool_Config_t

Data structure to customize the WLAN driver buffer pool parameters.

See also

qapi_WLAN_Set_Param
__QAPI_WLAN_PARAM_GROUP_SYSTEM_DRIVER_NETBUF_POOL_SIZE

Data fields

Type	Parameter	Description
uint32_t	pool_Size	Number of driver buffer objects the WLAN driver is to use for transmitting/receiving data packets over WLAN. This buffer pool is only for WLAN driver metadata and not for actual packet buffers. Modifying this parameter can have an impact on WLAN performance. The system default for this parameter is 10.
uint32_t	rx_Threshold	Number of driver buffer objects to reserve for receive. The system default for this parameter is 2.

9.1.2.38 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.

Type	Parameter	Description
uint8_t	pattern_Mask[__QAPI_WLAN_PATTERN_MASK]	Pattern mask.
uint8_t	pattern[__QAPI_WLAN_PATTERN_MAX_SIZE]	Pattern string.

9.1.2.39 struct qapi_WLAN_Delete_Pattern_t

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

Data fields

Type	Parameter	Description
uint8_t	pattern_Index	Pattern index.
uint8_t	header_Type	Header type.

9.1.2.40 struct qapi_WLAN_Change_Default_Filter_Action_t

Data structure to change the action for the default packet filter.

Data fields

Type	Parameter	Description
uint8_t	pattern_Action_Flag	Pattern action flag.
uint8_t	header_Type	Header type.

9.1.2.41 struct qapi_WLAN_Security_Wep_Key_Pair_Params_t

Data structure to set/get the WEP key by the key index.

The key can be a maximum of 13 bytes long. Each byte represents either a single ASCII character or two hex characters. The application is responsible for converting ASCII and hex characters into hex numbers, and vice versa, if needed.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_WEP_KEY_PAIR qapi_WLAN_Set_Param qapi_WLAN_Get_Param

Data fields

Type	Parameter	Description
uint32_t	key_Index	WEP key index for which the key is provided.
uint32_t	key_Length	Length of the WEP Key
int8_t *	key	WEP key; memory for this key is to be provided by the application.

9.1.2.42 struct qapi_WLAN_Security_8021x_Private_Key_t

Data structure to set the 802.1x private key filename and its password.

Both Private_Key_filename and Private_Key_Password are ASCII.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PRIVATE_KEY qapi_WLAN_Set_Param

Data fields

Type	Parameter	Description
char *	Private_Key_-filename	Point to address where stores the private key filename
char *	Private_Key_-Password	Point to address where stores the private key password

9.1.2.43 struct qapi_WLAN_Preferred_Network_Offload_Config_t

Data structure to enable/disable the preferred network offload (PNO) feature. This data structure also allows applications to configure some of the profile-independent global parameters of the PNO feature.

The application sets this parameter by invoking qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NETWORK_OFFLOAD_ENABLE as the command ID.

Dependencies

This configuration should be done before adding any PNO profiles.

Data fields

Type	Parameter	Description
uint16_t	max_Num_-Preferred_-Network_-Profiles	Maximum number of PNO profiles that the application requires. Note that this is used for memory allocation in the WLAN firmware. The application must ensure that this value is always less than 15.
uint32_t	fast_Scan_-Interval_In_Ms	PNO framework allows applications to perform PNO scans in two different intervals – fast scans and slow scans. Fast scans are performed with the period of this parameter from the time PNO is enabled to fast_Scan_Duration_In_Ms. The periodicity of this scan is provided in milliseconds.
uint32_t	fast_Scan_-Duration_In_-Ms	Total fast scan duration during which a scan is to be performed every fast_Scan_Interval_In_Ms. This duration always starts from the time PNO is enabled. The reasoning behind this fast scan logic is to perform aggressive PNO scans for some time to actively search for APs. If no matching profiles are found for fast scan duration, the PNO scan interval increases to slow_Scan_Interval_In_Ms to perform the scan at an increased interval, thereby saving power. The fast scan duration is provided in milliseconds.

Type	Parameter	Description
uint32_t	slow_Scan_-Interval_In_Ms	This parameter signifies the PNO scan interval after the fast scan duration in fast_Scan_Duration_In_Ms has expired. In order to make PNO more power efficient, applications should ensure that this parameter is expected to have a value greater than fast_Scan_Interval_In_Ms, and the input is given in milliseconds.
uint8_t	start_Network-_List_Offload	Enable/disable a switch for the PNO feature; 1: Enable PNO, 0: Disable PNO.

9.1.2.44 struct qapi_WLAN_Preferred_Network_Profile_t

Data structure to set an AP profile information to search for when enabling the PNO feature.

This data structure is used to set the profile by passing the required profile information to qapi_WLAN_Set-_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NETWORK_PROFILE as the command ID.

Data fields

Type	Parameter	Description
uint8_t	index	Index of the AP profile to be set. This value should start with value 0 (for the first profile) and be incremented as each profile is added. However, the applications must ensure that this value does not exceed (max_Num_Preferred_Network_Profiles - 1) whereas max_Num_Preferred_Network_Profiles is the parameter provided by the application when enabling the PNO feature.
uint8_t	ssid_Len	Length of the SSID of the AP profile that the application is looking for in the PNO scan.
uint8_t	ssid[__QAPI_-WLAN_MAX-_SSID_LEN]	SSID of the AP profile that the application is looking for in the PNO scan.
qapi_WLAN_-Auth_Mode_e	auth_Mode	Authentication mode of the AP profile.
qapi_WLAN_-Crypt_Type_e	encryption_-Type	Encryption type of the AP profile.

9.1.2.45 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_WIRE-LESS_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.

9.1.2.46 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.

9.1.2.47 struct qapi_WLAN_Set_PMKID_Params_t

Data structure that the application is to set pmkid.

Data fields

Type	Parameter	Description
uint8_t	bssid[__QAPI-_WLAN_MA-C_LEN]	bssid of the wlan connection.
uint8_t	enable	qapi_WLAN_PMKID_ENABLE_e.
uint8_t	pmkid[__QAP-I_WLAN_PM-KID_LEN]	pmkid of the wlan connection.

9.1.2.48 struct qapi_WLAN_RxEapolKey_Cb_Info_t

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

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

Data fields

Type	Parameter	Description
uint8_t	descType	Eapol key type, qapi_EAPOL_KEY_TYPE_e.
uint8_t	keyInfo[2]	Key information, big endian.
uint8_t	pmkid_valid	Is pmkid valid.
uint8_t	rsrv[4]	Reserved.
uint8_t	pmkid[__QAPI_WLAN_PMKID_LEN]	Pmkid.

9.1.2.49 struct qapi_WLAN_Set_Finite_Cyclic_Groups_Params_t

Data structure that presents SAE finite cyclic groups configured by application.

Data fields

Type	Parameter	Description
uint32_t *	sae_groups	SAE groups.
uint32_t	num	The number of configured SAE groups.

9.1.2.50 struct qapi_WLAN_SAE_PMK_Cache_t

Data structure that enables/disables Pairwise Master Key (PMK) cache function for SAE.

Data fields

Type	Parameter	Description
uint32_t	op	Cache operation.

9.1.2.51 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_LEN+1]	Key.

Type	Parameter	Description
uint8_t	key_Length	Key length.
uint8_t	mac_Addr[__QAPI_WLAN_MAC_LEN]	MAC address.

9.1.2.52 struct qapi_WLAN_WPS_Cb_Info_t

Data structure to get WPS command results from the driver.

Data fields

Type	Parameter	Description
uint8_t	status	WPS status.
uint8_t	error	WPS error.
qapi_WLAN_WPS_Credentials_t	credential	WPS credentials.
uint8_t	peer_dev_addr[__QAPI_WLAN_MAC_LEN]	MAC address of a peer device.

9.1.2.53 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.

9.1.2.54 struct qapi_WLAN_Scan_List_t

Data structure to store results of a scan.

Data fields

Type	Parameter	Description
uint32_t	num_Scan_-Entries	Number of scan results.
void *	scan_List	Scan results.

9.1.2.55 struct qapi_WLAN_5G_Prefer_t

Data structure to store the preferred 5G parameter.

Data fields

Type	Parameter	Description
uint8_t	is_prefer_5g	Enables preferred 5G function.
int8_t	rssi_hi_5g	RSSI value base in high region.
int8_t	rssi_mid_5g	RSSI value base in middle region.
int8_t	rssi_lo_5g	RSSI value base in low region.
int8_t	pref_hi_5g	Preferred value added in RSSI high region.
int8_t	pref_mid_5g	Preferred value added in RSSI middle region.
int8_t	pref_lo_5g	Preferred value added in RSSI low region.

9.1.2.56 struct qapi_WLAN_Connect_Policy_t

Data structure to store connection policy.

Data fields

Type	Parameter	Description
uint8_t	best_signal_-connection	Enables (1) or disables (0) the best signal connection feature.
uint8_t	rssi_threshold	Connection RSSI threshold value.
uint8_t	is_wpax_auth_-ignore	Sets connection without security type.
qapi_WLAN_-5G_Prefer_t	prefer_5g	Configures preferred 5G function settings
uint16_t	reserved	Reserved for future use.

9.1.2.57 struct qapi_WLAN_Set_BMcast_Filter_t

Stores the set broadcast and multicast.

Data fields

Type	Parameter	Description
uint8_t	enable	Enables (1) or disables (0) broadcast/multicast Rx filter.
uint8_t	reserved	Reserved for future use.

9.1.2.58 struct qapi_WLAN_Tx_Status_t

Data structure to get the Tx pipe status.

Data fields

Type	Parameter	Description
uint16_t	status	One of QAPI_WLAN_TX_STATUS_∗ types.

9.1.2.59 struct qapi_WLAN_Program_Mac_Addr_Param_t

Data structure to store data for setting the MAC address command.

Data fields

Type	Parameter	Description
uint8_t	addr[6]	MAC address.
uint8_t	result	Result of the command.

9.1.2.60 struct qapi_WLAN_Dbglog_Enable_t

Data structure to enable/disable logging in the target.

This data structure enables the DBGLOG feature and configures it with the default configuration for dbglogs. The default dbglog configuration enables the loglevel ERROR logs for all modules, configures a remote debug port, enables reporting, sets the report size to 0 (send to host only when the buffer is full), and sets the time resolution to zero.

Data fields

Type	Parameter	Description
qapi_WLAN_-Enable_e	enable	1:Enable, 0:Disable

9.1.2.61 struct qapi_WLAN_Dbglog_Module_Config_t

Data structure to modify the dbglog log level configuration for one or more modules in the target to capture the logs accordingly.

There are a total of 64 modules in the target to which the debug logs belong. Specify the module ID mask for which the log level is to be configured/modified. Update the log level buffer with log level mask information at the proper index and position for each of the modules in the module ID mask specified.

The log level mask is 4 bits in length and is as following:

Bit0 – INFO;, Bit1 – LOW; Bit2 – HIGH; Bit3 – ERROR.

Each module occupies a nibble space in the log level buffer to store its log level mask at the index calculated using its module ID.

Data fields

Type	Parameter	Description
uint64_t	module_Id_-Mask	Module ID mask to configure/modify the log level for intended modules.

Type	Parameter	Description
uint32_t	log_Level[__QAPI_WLAN_DBGLOG_LOGLEVEL_INFO_LEN]	Contains the log level information for the modules in the module ID mask specified with the intended log level for each module.

9.1.2.62 struct qapi_WLAN_Dbglog_Config_t

Data structure to modify the dbglog configuration parameters in the target to capture the logs accordingly.

Data fields

Type	Parameter	Description
qapi_WLAN_Dbglog_Port_e	debug_Port	Used to configure the debug port on which the dbglogs is to be sent in the target. The target can send the captured debug logs either on its UART (local) or to the host, which in turn sends the logs on its configured debug UART when connected to a PC that has an external tool (QDL) running, which can capture and parse the dbglogs.
qapi_WLAN_Enable_e	reporting_Enable	Used to control the reporting of debug logs to the host or not when the debug port is configured as a remote UART. When the remote debug port is enabled, the host can configure the target to report the logs to the host or not. If reporting is enabled, the target can send logs to host when the buffer is full or when the reporting size is met, otherwise the logs keep getting overwritten in the target dbglog buffer with the new logs until reporting is enabled by the host.
uint32_t	report_Trigger_Size_In_Bytes	Used to configure number of debug messages after which the firmware sends the dbglogs to the host. By default, the firmware waits until the buffer is full. When the remote debug port is enabled, by default, the logs are sent to the host when the dbglog buffer in the target becomes full. The host can configure the report size for the number of dbglog messages the target can send logs to the host. The minimun number of messages expected is atleast 15 to avoid frequent events to the host.
uint32_t	misc_Config	Miscellaneous configuration for wlan dbglog.

9.1.2.63 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.

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

9.1.2.64 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.

9.1.2.65 struct qapi_WLAN_Cipher_t

Data structure for cipher.

Data fields

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

9.1.2.66 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_LENGTH]	[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.

Type	Parameter	Description
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

9.1.2.67 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.

9.1.2.68 struct qapi_WLAN_IPv6_Addr_t

Data structure for the IPv6 address.

Data fields

Type	Parameter	Description
uint8_t	ip_Address[__QAPI_WLAN_IPV6_ADDR_LEN]	IPv6 address.

9.1.2.69 struct qapi_WLAN_ARP_Offload_Config_t

Data structure for ARP offload.

Data fields

Type	Parameter	Description
uint8_t	enable	Enable/disable.
uint8_t	target_IP[__QAPI_WLAN_IPV4_ADDR_LEN]	IPV4 addresses of the local node.
uint8_t	target_Mac[__QAPI_WLAN_MAC_LEN]	MAC address for this tuple; if not valid, the local MAC is used.

9.1.2.70 struct qapi_WLAN_NS_Offload_Config_t

Data structure for neighbor solicitation offload.

Data fields

Type	Parameter	Description
uint8_t	enable	Enable/disable.

Type	Parameter	Description
qapi_WLAN_IPv6_Addr_t	target_IP[__QAPI_WLAN_NSOFF_MAX_TARGET_IPS]	IPV6 WLAN firmware address of the local node.
uint8_t	target_Mac[__QAPI_WLAN_MAC_LEN]	MAC address for this tuple; if not valid, the local MAC is used.
qapi_WLAN_IPv6_Addr_t	solicitation_IP	Multicast source IP addresses for receiving solicitations.

9.1.2.71 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.

9.1.2.72 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	Tx power policy.

9.1.2.73 struct qapi_WLAN_Rx_Aggrx_Params_t

Data structure for Rx A-MPDU.

Data fields

Type	Parameter	Description
uint8_t	aggrx_Buffer_Size	Buffer size.
uint8_t	aggrx_Reorder_Buffer_Timeout_In_Ms	Buffer reorder timeout.
uint8_t	aggrx_Session_Timeout_Val_In_Ms	Session timeout.

Type	Parameter	Description
uint8_t	aggrx_Reorder-_Cfg	Reorder configuration.
uint8_t	aggrx_Session-_Timeout_Cfg	Session timeout.
uint8_t	reserved0	Reserved.

9.1.2.74 struct qapi_WLAN_BSS_Max_Idle_Period_t

Data structure to pass 802.11v BSS maximum idle period information from the application to the driver. Used only in SoftAP mode.

Data fields

Type	Parameter	Description
uint16_t	period	Period in terms of 1000 TUs.
uint16_t	protected_-Keep_Alive	If protected, keepalives are required.

9.1.2.75 struct qapi_WLAN_WNM_Sleep_Period_t

Data structure to pass 802.11v WNM sleep period information from the application to the driver. Used only in Station mode.

Data fields

Type	Parameter	Description
uint16_t	action_Type	Enter: 1, exit: 0.
uint16_t	duration	Duration in terms of DTIM intervals.

9.1.2.76 struct qapi_WLAN_WNM_Cb_Info_t

Data structure to get 802.11v event information from the driver.

Data fields

Type	Parameter	Description
uint16_t	cmd_Type	WNM command type.
uint16_t	response	WNM command result.

9.1.2.77 struct qapi_WLAN_Sta_Config_Bmiss_Config_t

Data structure to be used in case the application wants to change Beacon Miss parameters.

On receiving this request from a user in STA mode, the WLAN firmware reports a link loss after the specified interval. Either one of the values will be set by the user; bmiss_Time_In_Ms or num_Beacons. Based on the value set, the firmware detects a beacon miss and triggers a disconnect in STA mode.

Data fields

Type	Parameter	Description
uint16_t	bmiss_Time_-In_Ms	New beacon miss time (in ms) to be set by application.
uint16_t	num_Beacons	New number of beacons to be set by the application to detect the number of beacons missed, after which a disconnect is triggered.

9.1.2.78 struct qapi_WLAN_Hw_Ant_Div_Config_t

Data structure to configure the parameters of HW antenna diversity .

Data fields

Type	Parameter	Description
uint8_t	mode	Refer to enum qapi_WLAN_Ant_Div_Mode_e .
uint8_t	reserved1	Reserved for future use .
uint16_t	reserved2	Reserved for future use .
uint32_t	param	If mode is auto mode, it can be skipped, if mode is packet number strictly, it is packet number and if mode is time interval strictly, it is time interval .

9.1.2.79 struct qapi_WLAN_Ant_Div_Config_t

Data structure to configure the parameters of software antenna diversity .

Data structure to be used in case the application wants to configure HW antenna diversity.

Data fields

Type	Parameter	Description
uint8_t	enable_Ant_-Div	Refer to enum qapi_WLAN_Ant_Div_Enable_e.
uint8_t	tx_Follow_Rx	1: tx follows rx, 0: tx does not follow rx.
union qapi_W-LAN_Ant_Div-_Config_t	ant_div_config	

9.1.2.80 union qapi_WLAN_Ant_Div_Config_t.ant_div_config

Data fields

Type	Parameter	Description
qapi_WLAN_-Hw_Ant_Div_-Config_t	hw_ant_div_-config	ConfigureS hardware antenna diversity parameter .
qapi_WLAN_-Sw_Ant_Div_-Config_t	sw_ant_div_-config	Reserved for future use.

9.1.2.81 struct qapi_WLAN_Get_Ant_Div_t

Data structure used to retrieve the current antenna diversity information .

Data fields

Type	Parameter	Description
uint8_t	enable_Ant_-Div	refer to enum qapi_WLAN_Ant_Div_Enable_e.
uint8_t	tx_Follow_Rx	1: tx antenna follows rx antenna, 0:tx antenna not follows rx antenna.
uint8_t	curr_Rx_Ant_-2g	Current rx physical antenna in 2G band.
uint8_t	curr_Tx_Ant_-2g	Current tx physical antenna in 2G band.
uint8_t	curr_Rx_Ant_-5g	Current rx physical antenna in 5G band.
uint8_t	curr_Tx_Ant_-5g	Current tx physical antenna in 5G band.
uint8_t	avg_Main_Rssi	Current average main rssi.
uint8_t	avg_Alt_Rssi	Current average alternative rssi.
uint32_t	total_Pkt_Cnt	Packet count used to calculate the sum of rssi.
uint32_t	ant_Switch_Cnt	Number of switching antennas.

9.1.2.82 struct qapi_WLAN_Pmf_Evt_t

Data structure that receives the event related to WLAN PMF.

Data fields

Type	Parameter	Description
uint8_t	event_type	PMF event type that can be set to the value as qapi_WLAN_PMF_EVENT_TYPE_e listed.
uint8_t	reserved[3]	Reserved for alignment.
uint32_t	param	PMF event parameter.

9.1.2.83 struct qapi_WLAN_Gen_Pkt_Params_t

Data structure that the parameters are needed when invoking qapi_WLAN_Gen_Pkt() to generate packet.

Data fields

Type	Parameter	Description
uint8_t	peer_mac_-addr[__QAPI_-WLAN_MAC-_LEN]	Destination address.
uint16_t	type	Type of data frame.
uint32_t	counter	Counts the number of packets that will be sent (0 to infinity).
uint32_t	interval	Sends packets periodically (in milliseconds $\leq$ >= 100 ms).
uint32_t	data_len	Payload length, $\geq$ >= 1496 btyes.
uint8_t *	data	Payload.

9.1.2.84 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.
uint16_t	max_bw	Maximum bandwidth

9.1.2.85 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.

9.1.2.86 struct qapi_WLAN_Get_Power_Evt_t

Data fields

Type	Parameter	Description
uint8_t	reg_power	
uint8_t	ctl_power	
uint16_t	target_power	
uint16_t	real_power	

9.1.2.87 struct qapi_WLAN_Set_Rate_Params_t

Sets the rate.

Data fields

Type	Parameter	Description
uint8_t	ra_ON	
uint8_t	rate_staid	
uint8_t	rate_p_rate	

Type	Parameter	Description
uint8_t	rate_s_rate	
uint8_t	rate_t_rate	

9.1.2.88 struct qapi_WLAN_Listen_Interval_Params_t

Sets the STA listen interval.

Data fields

Type	Parameter	Description
uint32_t	time	Interval time.
uint32_t	round_type	Round type.

9.1.2.89 struct qapi_WLAN_Edca_Params_t

Sets the STA edca parameters, including aifsn/cw_min/cw_max/txoplimit.

Data fields

Type	Parameter	Description
uint8_t	qid	
uint8_t	aifsn	
uint16_t	cw_min	
uint16_t	cw_max	
uint16_t	txop_limit	

9.1.2.90 struct qapi_WLAN_BA_Window_Params_t

Sets the STA BA window size.

Data fields

Type	Parameter	Description
uint16_t	ack_timeout	
uint16_t	delay	

9.1.3 Typedef Documentation

9.1.3.1 typedef void(∗ qapi_WLAN_Callback_t)(uint8_t device_ID, uint32_t cb_ID, void ∗application_Context, void ∗payload, uint32_t payload_Length)

Function pointer to an application specified callback handler function.

The callback handler for WLAN commands can be set using qapi_WLAN_Set_Callback().

Parameters

in	*device_ID*	Device ID.
in	*cb_ID*	Callback ID associated to a command.
in	*application_Context*	Application context.
in	*payload*	Data.

in	*payload_Length*	Data size.

Returns

None.

9.1.4 Enumeration Type Documentation

9.1.4.1 enum qapi_WLAN_Enable_e

Identifies the enable/disable options for WLAN.

Enumerator:

QAPI_WLAN_DISABLE_E Disable the Wi-Fi module.
QAPI_WLAN_ENABLE_E Enable the Wi-Fi module.

9.1.4.2 enum qapi_WLAN_Scan_Ctrl_Flag_bits_e

Enum declarations for the scan control flags.

NOTE The ScanCtrlFlag value of 0xFF is used to disable all flags in the Scan Params Cmd. Do not add any more flags to qapi_WLAN_Scan_Ctrl_Flag_bits_e.

Enumerator:

QAPI_WLAN_CONNECT_SCAN_CTRL_FLAGS_E Set if the Connect command can be scanned.
QAPI_WLAN_SCAN_CONNECTED_CTRL_FLAGS_E Set if the scan is for the SSID it is already connected to.
QAPI_WLAN_ACTIVE_SCAN_CTRL_FLAGS_E Set if active scan is enabled.
QAPI_WLAN_ROAM_SCAN_CTRL_FLAGS_E Set if roam scan is enabled when BMISS and LOWRSSI.
QAPI_WLAN_REPORT_BSSINFO_CTRL_FLAGS_E Set if the scan follows the customer's BSSINFO reporting rule.
QAPI_WLAN_ENABLE_AUTO_CTRL_FLAGS_E If disabled, the target does not scan after a disconnect event.
QAPI_WLAN_ENABLE_SCAN_ABORT_EVENT_E Scan complete event with cancelled status will be generated when a scan is pre-empted before it is completed.
QAPI_WLAN_ENABLE_DFS_SKIP_CTRL_FLAGS_E Set to skip scanning a DFS channel.

9.1.4.3 enum qapi_WLAN_Scan_Status_e

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

9.1.4.4 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.

9.1.4.5 enum qapi_WLAN_Filterable_Event_e

Data structure that enumerates the list of WLAN subsystem events, which the application can chose to filter, thereby avoiding application processor wakeups because of these events.

Enumerator:

QAPI_WLAN_BSSINFO_EVENTID_E Event ID to filter the BSS Info event sent by the WLAN firmware.
QAPI_WLAN_EXTENSION_EVENTID_E Event ID to filter the WMI Extension events sent by the WLAN firmware; applicable for device-0 only.
QAPI_WLAN_CHANNEL_CHANGE_EVENTID_E Event ID to filter the channel change event sent by the WLAN firmware.
QAPI_WLAN_PEER_NODE_EVENTID_E Event ID to filter the events sent by the WLAN firmware during an IBSS connection; applicable for device-0 only.
QAPI_WLAN_ADDBA_REQ_EVENTID_E Event ID to filter the event sent by the WLAN firmware notifying the host that it has sent an ADDBA request for a BlockACK session.
QAPI_WLAN_ADDBA_RESP_EVENTID_E Event ID to filter the event sent by the WLAN firmware indicating the reception of an ADDBA response for the ADDBA request sent.
QAPI_WLAN_DELBA_REQ_EVENTID_E Event ID to filter a DELBA request event.
QAPI_WLAN_P2P_INVITE_REQ_EVENTID_E Event ID to filter the events from the WLAN firmware when a P2P invitation request is received from a device; applicable for device-0 only.
QAPI_WLAN_P2P_INVITE_RCVD_RESULT_EVENTID_E Event ID to filter the events from the WLAN firmware when a P2P invitation response is sent to a peer device; applicable for device-0 only
QAPI_WLAN_P2P_INVITE_SENT_RESULT_EVENTID_E Event ID to filter the events from the WLAN firmware when a P2P invitation response is received from a peer device; applicable for device-0 only.
QAPI_WLAN_P2P_PROV_DISC_RESP_EVENTID_E Event ID to filter the events from the WLAN firmware when a P2P provision discovery response is received from a peer device; applicable for device-0 only.
QAPI_WLAN_P2P_PROV_DISC_REQ_EVENTID_E Event ID to filter the events from the WLAN firmware when a P2P provision discovery request is received from a peer device; applicable for device-0 only.
QAPI_WLAN_P2P_START_SDPD_EVENTID_E Event ID to filter a P2P service discovery, provision discovery start events; applicable for device-0 only.
QAPI_WLAN_P2P_SDPD_RX_EVENTID_E Event ID to filter a P2P service discovery, provision discovery Rx events; applicable for device-0 only.
QAPI_WLAN_DBGLOG_EVENTID_E Event ID to filter the DBGLOG event sent by the WLAN firmware when the dbglog buffer is full or the threshold is reached; applicable for device-0 only.
QAPI_WLAN_PKTLOG_EVENTID_E Event ID to filter the PKTLOG event sent by the WLAN firmware when the pktlog buffer is full; applicable for device-0 only.
QAPI_WLAN_WPS_PROFILE_EVENTID_E Event ID to filter events sent by the WLAN firmware when a WPS profile is received from a successful WPS handhshake.
QAPI_WLAN_FLOW_CONTROL_EVENTID_E Event ID to filter the bus flow control event sent by the WLAN firmware.
QAPI_WLAN_P2P_REQ_TO_AUTH_EVENTID_E Event ID to filter an event indicating pending authentication for a P2P group owner negotiation request received from a peer device; applicable for device-0 only.

QAPI_WLAN_DIAGNOSTIC_EVENTID_E Event ID to filter diagnostic events sent by the WLAN firmware; applicable for device-0 only.
QAPI_WLAN_WNM_EVENTID_E Event ID to filter WNM events sent by the WLAN firmware.

9.1.4.6 enum qapi_WLAN_Wait_For_Status_e

Enumeration that provides various blocking options for qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param() APIs when invoked by the application.

Enumerator:

QAPI_WLAN_NO_WAIT_E Makes a QAPI nonblocking.
QAPI_WLAN_WAIT_E Makes a QAPI blocking.

9.1.4.7 enum qapi_WLAN_Dev_Mode_e

Enumeration that provides a list of supported operating modes for each virtual device.

Enumerator:

QAPI_WLAN_DEV_MODE_STATION_E Infrastructure non-AP Station mode, supported in both Single Device mode and Concurrent mode. When operating in Concurrent mode, virtual device 1 must be used for the Station mode of operation.
QAPI_WLAN_DEV_MODE_AP_E Soft-AP mode Infrastructure AP Station mode, supported in both Single Device mode and Concurrent mode. When operating in Concurrent mode, virtual device 0 must be used for the AP mode of operation.
QAPI_WLAN_DEV_MODE_INVALID_E Independent BSS mode of operation, supported in Single Device mode only. Virtual device 0 must be used for the IBSS mode of operation. Invalid device mode.

9.1.4.8 enum qapi_WLAN_Phy_Mode_e

Enumeration that provides a list of supported 802.11 PHY modes.

Enumerator:

QAPI_WLAN_11B_MODE_E 802.11b.
QAPI_WLAN_11G_MODE_E 802.11g.
QAPI_WLAN_11NG_HT20_MODE_E 802.11b/g/n HT20.
QAPI_WLAN_11A_MODE_E 802.11a.
QAPI_WLAN_11A_HT20_MODE_E 802.11a with HT20.
QAPI_WLAN_11ABGN_HT20_MODE_E 802.11 CCX.

9.1.4.9 enum qapi_WLAN_11n_HT_Config_e

Enumeration that provides 11n HT configurations.

Enumerator:

QAPI_WLAN_11N_DISABLED_E 802.11n disabled.
QAPI_WLAN_11N_HT20_E 802.11n with bandwith 20M.
QAPI_WLAN_11N_HT40_E 802.11n with bandwith 40M.

9.1.4.10 enum qapi_WLAN_Power_Mode_e

Enumeration of a list of supported power modes in the WLAN subsystem.

Enumerator:

QAPI_WLAN_POWER_MODE_REC_POWER_E Power Save mode that enables both MAC and system power save.

QAPI_WLAN_POWER_MODE_MAX_PERF_E Maximum performance mode that disables both MAC and system power save.

9.1.4.11 enum qapi_WLAN_Power_Module_e

Enumeration of various WLAN modules that have the ability to request the necessary power mode for their operation. Each of the modules here has the ability to request Power Save mode or Maximum Performance mode. This level of granularity mainly helps to debug mismatches between Operating Power mode and Expected Power mode of the WLAN subsystem.

Enumerator:

QAPI_WLAN_POWER_MODULE_USER_E If the application requires the system to be in Maximum Performance mode.

QAPI_WLAN_POWER_MODULE_WPS_E WPS handshakes should be in Maximum Performance mode when the application initiates a WPS handshake. After a WPS handshake is completed, the application can request Power Save mode with the source module as WPS.

QAPI_WLAN_POWER_MODULE_P2P_E Power mode set by the P2P module. P2P handshakes should be in Maximum Performance mode when the application initiates P2P control commands. After a P2P session is established (Group Owner or Client mode), if the device operates in GO mode, the application is expected to request Maximum Perfmance mode with the module owner as P2P and is expected to be in Maximum Performance mode until the P2P session is active. If the device operates in P2P Client mode, the application can optionally request Power Save mode with the module owner as P2P once the connection is established.

QAPI_WLAN_POWER_MODULE_SOFTAP_E Power mode set by SoftAP module. Applications must operate in Maximum Performance mode as long as the device is operating in SoftAP mode.

QAPI_WLAN_POWER_MODULE_SRCALL_E Power mode request by the store recall module. At this point, the WLAN driver internally uses this and the application is not expected to use it.

QAPI_WLAN_POWER_MODULE_TKIP_E Power mode set by the TKIP countermeasure module. At this point, the WLAN driver internally uses this and the application is not expected to use it.

QAPI_WLAN_POWER_MODULE_RAW_MODE_E Power mode request for sending packets in Raw mode. This is the mode where the application constructs and transmits packets over the WLAN radio, bypassing the IP stack and WLAN MLME layer. Applications must request Maximum Performance before initiating any Raw mode packet transmissions.

QAPI_WLAN_POWER_MODULE_PWR_MAX_E Maximum value for modules

9.1.4.12 enum qapi_WLAN_DTIM_Policy_e

Enumeration of supported DTIM policies.

Enumerator:

QAPI_WLAN_DTIM_IGNORE_E Non-AP station wakeups only at TIM intervals; ignores content after beacon (CAB).

QAPI_WLAN_DTIM_NORMAL_E Non-AP station wakeups at TIM intervals.

QAPI_WLAN_DTIM_STICK_E Non-AP station wakeups only at DTIM intervals.
QAPI_WLAN_DTIM_AUTO_E Non-AP station wakeups at both TIM and DTIM intervals.
QAPI_WLAN_DTIM_XSTICK_E Non-AP station wakeups only at X DTIM intervals.

9.1.4.13 enum qapi_WLAN_TX_Wakeup_Policy_e

Enumeration that identifies Tx wake-up policy.

Enumerator:

QAPI_WLAN_TX_WAKEUP_ON_SLEEP_E Transmit packets can exit WLAN from the protocol power save after transmitting 'n' number of packets in Power Save mode. The value of 'n' can be programmed through qapi_WLAN_Power_Policy_Params_t.
QAPI_WLAN_TX_WAKEUP_SKIP_ON_SLEEP_E Transmit packets need not exit WLAN from protocol power save irrespective of the number of packets transmitted while in Power Save mode.

9.1.4.14 enum qapi_WLAN_Power_Save_Policy_e

Enumeration that identifies the WLAN power save event policy.

Enumerator:

QAPI_WLAN_PS_SEND_POWER_SAVE_FAIL_EVENT_ALWAYS_E Firmware should indicate NULL frame transmission failures when entering and exiting protocol power save (PM bit transitions) through target events.
QAPI_WLAN_PS_IGNORE_POWER_SAVE_FAIL_EVENT_DURING_SCAN_E Firmware can ignore the NULL frame transmission failures when entering and exiting protocol power save (PM bit transitions).

9.1.4.15 enum qapi_WLAN_IP_Protocol_Type_e

Enumeration of various IP versions, which is used for the TCP Keepalive offload feature.

Enumerator:

QAPI_WLAN_IP_PROTOCOL_V4_E Used to indicate IPv4 protocol.
QAPI_WLAN_IP_PROTOCOL_V6_E Used to indicate IPv6 protocol.

9.1.4.16 enum qapi_WLAN_Pkt_Filter_Type_e

Enumeration of header types supported by the WLAN packet filtering feature..

Enumerator:

QAPI_WLAN_PKT_FILTER_MAC_HEADER_E Used to configure the WOW/packet filter pattern for the MAC header packet.
QAPI_WLAN_PKT_FILTER_SNAP_E Used to configure the WOW/packet filter pattern for the SNAP header packet.
QAPI_WLAN_PKT_FILTER_ARP_E Used to configure the WOW/packet filter pattern for the ARP packet.
QAPI_WLAN_PKT_FILTER_IPV4_E Used to configure the WOW/packet filter pattern for the IPV4 packet.
QAPI_WLAN_PKT_FILTER_IPV6_E Used to configure the WOW/packet filter pattern for the IPV6 packet.

QAPI_WLAN_PKT_FILTER_ICMP_E Used to configure the WOW/packet filter pattern for the ICMP packet.

QAPI_WLAN_PKT_FILTER_IGMP_E Used to configure the WOW/packet filter pattern for the IGMP packet.

QAPI_WLAN_PKT_FILTER_ICMPV6_E Used to configure the WOW/packet filter pattern for the ICMPV6 packet.

QAPI_WLAN_PKT_FILTER_UDP_E Used to configure the WOW/packet filter pattern for the UDP packet.

QAPI_WLAN_PKT_FITLER_TCP_E Used to configure the WOW/packet filter pattern for the TCP packet.

QAPI_WLAN_PKT_FITLER_PAYLOAD_E Used to configure the WOW/packet filter pattern for the payload.

9.1.4.17 enum qapi_WLAN_Auth_Mode_e

Enumeration that idetifies authentication modes supported by the WLAN subsystem. The application sets the required authentication from one of these modes using qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_SECURITY_AUTH_MODE as the command ID.

Dependencies

Authentication mode should be set before calling qapi_WLAN_Commit() to make the authentication mode set effective.

Enumerator:

QAPI_WLAN_AUTH_NONE_E Open mode authentication.

QAPI_WLAN_AUTH_WPA_E Wi-Fi Protected Access v1 Protocol, if a preshared key (PSK) is not used (currently not supported).

QAPI_WLAN_AUTH_WPA2_E Wi-Fi Protected Access v2 Protocol, if a PSK is not used (currently not supported).

QAPI_WLAN_AUTH_WPA_PSK_E Wi-Fi Protected Access v1, if a PSK is used.

QAPI_WLAN_AUTH_WPA2_PSK_E Wi-Fi Protected Access v2, if a PSK is used.

QAPI_WLAN_AUTH_WPA_CCKM_E WPA v1 with Cisco Centralized Key Management (currently not supported).

QAPI_WLAN_AUTH_WPA2_CCKM_E WPA v2 with Cisco Centralized Key Management (currently not supported)

QAPI_WLAN_AUTH_WPA2_PSK_SHA256_E WPA2-PSK using SHA256 (currently not supported).

QAPI_WLAN_AUTH_WEP_E Wired Equivalent Privacy (WEP) mode of authentication.

QAPI_WLAN_AUTH_WPA3_SAE_E WPA3 SAE mode of authentication.

QAPI_WLAN_AUTH_WPA2_SAE_MIXED_E WPA2 and SAE mixed mode of authentication.

QAPI_WLAN_AUTH_INVALID_E Invalid authentication method.

9.1.4.18 enum qapi_WLAN_Crypt_Type_e

Enumeration that identifies a list of supported encryption methods. The application sets the required encryption from one of these modes using qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_SECURITY_AUTH_MODE as the command ID.

Dependencies

Encryption mode should be set before calling qapi_WLAN_Commit() to make the authentication mode set effective.

Enumerator:

QAPI_WLAN_CRYPT_NONE_E No encryption, Open mode.
QAPI_WLAN_CRYPT_WEP_CRYPT_E WEP mode of encryption.
QAPI_WLAN_CRYPT_TKIP_CRYPT_E Temporal Key Integrity Protocol (TKIP).
QAPI_WLAN_CRYPT_AES_CRYPT_E Advanced Encryption Standard (AES).
QAPI_WLAN_CRYPT_WAPI_CRYPT_E WLAN Authentication and Privacy Infrastructure; currently not supported.
QAPI_WLAN_CRYPT_BIP_CRYPT_E Broadcast Integrity Protocol; currently not supported.
QAPI_WLAN_CRYPT_KTK_CRYPT_E Key Transport Key; currently not supported.
QAPI_WLAN_CRYPT_INVALID_E Invalid encryption type.

9.1.4.19 enum qapi_WLAN_8021X_Method_e

Enumeration that identifies a list of supported 802.1x methods. The application sets the required method from one of these modes using qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_METHOD as the command ID.

Dependencies

802.1x method should be set before calling qapi_WLAN_Commit() to make the method set effective.

Enumerator:

QAPI_WLAN_8021X_METHOD_UNKNOWN Unknown.
QAPI_WLAN_8021X_METHOD_EAP_TLS_E EAP_TLS.
QAPI_WLAN_8021X_METHOD_EAP_TTLS_MSCHAPV2_E EAP_TTLS_MSCHAPV2.
QAPI_WLAN_8021X_METHOD_EAP_PEAP_MSCHAPV2_E EAP_PEAP_MSCHAPV2.
QAPI_WLAN_8021X_METHOD_EAP_TTLS_MD5_E EAP_TTLS_MD5.
QAPI_WLAN_8021X_METHOD_MAX max.

9.1.4.20 enum qapi_WLAN_Raw_Mode_Header_Type_e

Enumeration of a list of supported 802.11 header types for a Raw mode transmission.

Enumerator:

QAPI_WLAN_RAW_MODE_HDR_TYPE_BEACON_E Raw mode frame header is of type beacon.
QAPI_WLAN_RAW_MODE_HDR_TYPE_PROBE_REQ_DATA_E Raw mode frame header is of type probe request.

QAPI_WLAN_RAW_MODE_HDR_TYPE_QOS_DATA_E Raw mode frame header is of type QOS data.
QAPI_WLAN_RAW_MODE_HDR_TYPE_FOUR_ADDR_DATA_E Raw mode frame header is of type 4 address data.
QAPI_WLAN_RAW_MODE_HDR_TYPE_USER_DEFINED_E Raw mode frame header is of type self-defined.

9.1.4.21 enum qapi_WLAN_PMKID_ENABLE_e

Enumeration the enable/disable options for WLAN PMKID.

Enumerator:

QAPI_WLAN_PMKID_DISABLE_E Disable the WLAN PMKID.
QAPI_WLAN_PMKID_ENABLE_E Enable the WLAN PMKID.

9.1.4.22 enum qapi_EAPOL_KEY_TYPE_e

Enumeration that provides various EAPOL key type for qapi_WLAN_RxEapolKey_Cb_Info_t.

Enumerator:

QAPI_EAPOL_KEY_TYPE_WPA2 WPA2 key type.
QAPI_EAPOL_KEY_TYPE_WPA WPA key type.

9.1.4.23 enum qapi_WLAN_WPS_Connect_Action_e

Enumeration that identifies the action to be taken after a WPS initial request succeeds.

Enumerator:

QAPI_WLAN_WPS_NO_ACTION_POST_CONNECT_E No action to be taken after WPS initiation succeeds.
QAPI_WLAN_WPS_CONNECT_REGISTRAR_IN_ENROLLEE_E Set up a connection after WPS initiation succeeds.

9.1.4.24 enum qapi_WLAN_WPS_Mode_e

Enumeration of supported WPS modes.

Enumerator:

QAPI_WLAN_WPS_PIN_MODE_E WPS Pushbutton method.
QAPI_WLAN_WPS_PBC_MODE_E WPS PIN method.

9.1.4.25 enum qapi_WLAN_WPS_Status_Code_e

Enumeration that identifies the types of WPS command results.

Enumerator:

QAPI_WLAN_WPS_STATUS_SUCCESS WPS succeeded.
QAPI_WLAN_WPS_STATUS_FAILURE WPS failed.
QAPI_WLAN_WPS_STATUS_IDLE WPS in the Idle state.
QAPI_WLAN_WPS_STATUS_IN_PROGRESS WPS in progress.

9.1.4.26 enum qapi_WLAN_WPS_Error_Code_e

Enumeration that identifies results of WPS pushbutton/WPS PIN operations.

Enumerator:

QAPI_WLAN_WPS_ERROR_SUCCESS_E WPS initiation succeeded.
QAPI_WLAN_WPS_ERROR_INVALID_START_INFO_E Invalid information given to initialize WPS.
QAPI_WLAN_WPS_ERROR_MULTIPLE_PBC_SESSIONS_E Multiple WPS pushbutton sessions.
QAPI_WLAN_WPS_ERROR_WALKTIMER_TIMEOUT_E WPS walktimer expired.
QAPI_WLAN_WPS_ERROR_M2D_RCVD_E M2D message received.
QAPI_WLAN_WPS_ERROR_PWD_AUTH_FAIL_E Authentication failed.
QAPI_WLAN_WPS_ERROR_CANCELLED_E WPS cancelled.
QAPI_WLAN_WPS_ERROR_INVALID_PIN_E Incorrect WPS PIN.

9.1.4.27 enum qapi_WLAN_Mac_Result_e

Enum that identifies the results of the command to set the MAC address.

Enumerator:

QAPI_WLAN_SET_MAC_RESULT_SUCCESS_E Successfully (re)programmed the MAC address into the Wi-Fi device.
QAPI_WLAN_SET_MAC_RESULT_DEV_DENIED_E Device denied the operation for several possible reasons, the most of which is that the MAC address equals the current MAC address that is already in the device. An invalid MAC address value can also cause this result.
QAPI_WLAN_SET_MAC_RESULT_DEV_FAILED_E Device tried but failed to program the MAC address. An error occurred on the device as it tried to program the MAC address.
QAPI_WLAN_SET_MAC_RESULT_DRIVER_FAILED_E Driver tried but failed to program the MAC address. Possibly, the driver did not have the proper code compiled to perform this operation.

9.1.4.28 enum qapi_WLAN_Dbglog_Port_e

Enumeration to configure the debug port for the debug logs in the firmware.

The target can send the captured debug logs either on its UART (local) or to the host, which in turn sends on its configured debug UART when connected to a PC, where an external tool (QDL) that is running can capture and parse the dbglogs.

Enumerator:

QAPI_WLAN_DBGLOG_LOCAL_PORT_E Used when logs are needed on the target (Kingfisher) UART.
QAPI_WLAN_DBGLOG_REMOTE_PORT_E Used when logs are needed on a host that is connected to a PC running an external tool (QDL).

9.1.4.29 enum qapi_WLAN_Mode_e

Enumeration that identifies the device mode.

Enumerator:

MODE_STATION_E Station mode.
MODE_AP_E SoftAP mode supported only on device 0.
MODE_ADHOC_E Adhoc network; supported only on device 0.

MODE_MAXIMUM_E Maximum value for the device mode.

9.1.4.30 enum qapi_WLAN_DEV_Mode_e

Enumeration that identifies the device concurrency mode.

Enumerator:

DEV_MODE_STATION_E Station mode
DEV_MODE_AP_E SoftAP mode
DEV_MODE_AP_STA_E AP_STA Concurrency
DEV_MODE_NO_CONC_E Concurrency Off.

9.1.4.31 enum qapi_WLAN_Mgmt_Frame_Type_e

Enum declaration for management frame types.

Enumerator:

QAPI_WLAN_FRAME_BEACON_E Beacon frame type.
QAPI_WLAN_FRAME_PROBE_REQ_E Probe request frame type.
QAPI_WLAN_FRAME_PROBE_RESP_E Probe response frame type.
QAPI_WLAN_FRAME_ASSOC_REQ_E Association request frame type
QAPI_WLAN_FRAME_ASSOC_RESP_E Association response frame type.
QAPI_WLAN_NUM_MGMT_FRAME_E Number of management frame types.

9.1.4.32 enum qapi_WLAN_TX_Power_Policy_e

Enum declaration for Tx Power setting policy.

Enumerator:

QAPI_WLAN_POLLICY_SECURITY_E Security policy.
QAPI_WLAN_POLICY_NUM_E Number of policy.

9.1.4.33 enum qapi_WLAN_Disconnect_Reason_t

Identifies the disconnection reason.

Enumerator:

QAPI_WLAN_NO_NETWORK_AVAIL_E No network available.
QAPI_WLAN_LOST_LINK_E Missed beacons.
QAPI_WLAN_DISCONNECT_CMD_E User disconnect command.
QAPI_WLAN_BSS_DISCONNECTED_E BSS disconnected.
QAPI_WLAN_AUTH_FAILED_E Authentication failed.
QAPI_WLAN_ASSOC_FAILED_E Association failed.
QAPI_WLAN_NO_RESOURCES_AVAIL_E No resources available.
QAPI_WLAN_CSERV_DISCONNECT_E Disconnection due to connection services.
QAPI_WLAN_INVALID_PROFILE_E RSNA failure.
QAPI_WLAN_DOT11H_CHANNEL_SWITCH_E 802.11h channel switch.
QAPI_WLAN_PROFILE_MISMATCH_E Profile mismatched.
QAPI_WLAN_CONNECTION_EVICTED_E Connection evicted.
QAPI_WLAN_IBSS_MERGE_E Disconnection due to merging of IBSS.

QAPI_WLAN_EXCESS_TX_RETRY_E Tx frames failed after excessive retries.
QAPI_WLAN_SEC_HS_TO_RECV_M1_E Unused. Security 4-way handshake timed out waiting for M1.
QAPI_WLAN_SEC_HS_TO_RECV_M3_E Unused. Security 4-way handshake timed out waiting for M3.
QAPI_WLAN_TKIP_COUNTERMEASURES_E TKIP counter-measures.
QAPI_WLAN_CCX_TARGET_ROAMING_INDICATION_E Unused.
QAPI_WLAN_CCKM_ROAMING_INDICATION_E Unused.

9.1.4.34 enum qapi_WLAN_Ant_Div_Mode_e

Enumerate antenna switch based on which mode.

Enumerator:

QAPI_WLAN_ANT_DIV_PACKET_NUMBER_STRICTLY_E Antenna siwtch based on auto mode. In this mode, we switch antenna immediately once it is neccessary to switch antenna. For example, the main rssi is very low and the alt rssi is strong
QAPI_WLAN_ANT_DIV_TIME_INTERVAL_STRICTLY_E Antenna switch based on packet number strictly.

9.1.4.35 enum qapi_WLAN_Ant_Div_Enable_e

Enumerate antenna diversity enable mode.

Enumerator:

QAPI_WLAN_ENABLE_HW_ANT_DIV_E Disable ant div.
QAPI_WLAN_ENABLE_SW_ANT_DIV_E enable hw ant div.

9.1.4.36 enum qapi_WLAN_PMF_EVENT_TYPE_e

Enumerates PMF event type.

Enumerator:

QAPI_WLAN_PMF_SECURITY_MODE_VIOLATION_E DUT receives associate response with status code=30 and has comeback time IE.
QAPI_WLAN_PMF_POLICY_VIOLATION_E PMF and security mode is violated.

9.1.4.37 enum qapi_WLAN_PMF_CAP_e

Enumeration PMF capability.

Enumerator:

QAPI_WLAN_PMF_DISABLE_E WLAN PMF is disabled.
QAPI_WLAN_PMF_OPTIONAL_E WLAN PMF is optional.
QAPI_WLAN_PMF_REQUIRED_E WLAN PMF is required.

9.1.4.38 enum qapi_WLAN_RSN_CAP_e

Enumeration RSN capability.

Enumerator:

QAPI_WLAN_RSN_MFPR_E MFPR in RSN IE capability field.
QAPI_WLAN_RSN_MFPC_E MFPC in RSN IE capability field.

9.1.4.39 enum qapi_WLAN_Wpa3_Enable_e

_wpa3 Identifies the enable/disable options for WLAN wpa3.

Enumerator:

QAPI_WLAN_WPA3_DISABLE_E Disables WLAN wpa3.
QAPI_WLAN_WPA3_ENABLE_E Enables the WLAN wpa3.

9.1.4.40 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.

9.1.4.41 enum qapi_WLAN_FTM_CMD_e

Enum values for the FTM mode command type.

Enumerator:

QAPI_WLAN_FTM_SEND_REQ Send a packet request.
QAPI_WLAN_FTM_LOAD_BD Load board data.

9.1.5 Function Documentation

9.1.5.1 qapi_Status_t qapi_WLAN_Enable (qapi_WLAN_Enable_e *enable*)

Enables/disables the Wi-Fi module. This is a blocking call and returns on receipt of a WMI_READY event from KF.

Use QAPI_WLAN_ENABLE_E as the parameter for enabling and QAPI_WLAN_DISABLE_E for disabling WLAN.

This API brings up the KF firmware but does not add any devices.

Associated data types

qapi_WLAN_Enable_e

Parameters

in	*enable*	QAPI_WLAN_ENABLE_E or QAPI_WLAN_DISABLE_E.

See also

qapi_WLAN_Add_Device

Returns

QAPI_OK – Enabling or disabling WLAN succeeded.
QAPI_BUSY – Disabling WLAN failed due to open AF_INET (IPv4) sockets.
Nonzero value – Enabling or disabling WLAN failed.

Dependencies

Use qapi_WLAN_Add_Device() after qapi_WLAN_Enable() to add devices before using other WLAN control commands.
Use qapi_WLAN_Remove_Device() before disabling WLAN.

9.1.5.2 qapi_Status_t qapi_WLAN_Add_Device (uint8_t *device_ID*)

Adds an interface in the Wi-Fi driver. This API has no interaction with KF.

WLAN must be enabled before calling this API. A maximum of two devices are supported at this time.

Parameters

in	*device_ID*	Device ID.

Returns

QAPI_OK – Adding a new device (interface) succeeded.
Nonzero value – Adding a new device (interface) failed.

Dependencies

Use qapi_WLAN_Enable() to enable the Wi-Fi module before using this API.

9.1.5.3 qapi_Status_t qapi_WLAN_Start_Scan (uint8_t *device_ID,* const qapi_WLAN-_Start_Scan_Params_t * *scan_Params*)

Initiates a wireless scan to fnid nearby access points.

Users can configure the scan type as they want. Additional scan parameters can be tuned by using qapi_WLAN_Set_Params() with GroupID for Wireless and ParamID scan_params. Scaned results will be notified by QAPI_WLAN_SCAN_COMPLETE_CB_E

Associated data types

qapi_WLAN_Start_Scan_Params_t
#qapi_WLAN_Store_Scan_Results_e

Parameters

in	*device_ID*	Device ID.
in	*scan_Params*	scan parameters.

Returns

QAPI_OK – Wireless scan started.
Nonzero value – Wireless scan failed.

Dependencies

Scan cannot be started in SoftAP mode.

9.1.5.4 qapi_Status_t qapi_WLAN_Get_Scan_Results (uint8_t *device_ID,* qapi_WLAN_Scan_Comp_Evt_t * *scan_Res,* int16_t * *num_Bss*)

Returns the scan results from the most recent wireless scan performed to find nearby access points.

This is a blocking API and should be used when the QAPI_WLAN_BUFFER_SCAN_RESULTS_BLOCKING option is used.

Associated data types

qapi_WLAN_BSS_Scan_Info_t

Parameters

in	*device_ID*	Device ID.
out	*scan_Res*	Data pointer to get the scan result list from the driver.
in	*num_Bss*	Number of bss which can be put into

Returns

QAPI_OK – Getting the results from most recent wireless scan succeeded.
Nonzero value – Getting the results from most recent wireless scan failed.

Dependencies

Call qapi_WLAN_Start_Scan() to start a wireless scan before calling this API.

9.1.5.5 qapi_Status_t qapi_WLAN_Commit (uint8_t *device_ID*)

This API is part of connect/disconnect process in both non-AP Station and SoftAP modes. It uses previously set SSID and security configurations.

In non-AP Station mode, it allows the station to connect to or disconnect from an associated AP. In SoftAP mode, it starts the device as a SoftAP.

This API is already called from qapi_WLAN_Disconnect() and qapi_WLAN_WPS_Connect(), so it does not need to be called explicitly for these operations; however for connect operations (except for WPS), it will need to be called explicitly.

Parameters

in	*device_ID*	Device ID.

Returns

QAPI_OK – Commit operation succeeded.
Nonzero value – Commit operation failed.

Dependencies

SSID and security configurations must be set using qapi_WLAN_Set_Param() before calling this API. If SSID is set to a nonempty string before calling this API, it is considered as a connect request, and if the SSID is set to an empty string, it is considered as a disconnect request in both non-AP Station and SoftAP modes.
If the user wants to use Tx/Rx aggregation, __QAPI_WLAN_PARAM_GROUP_WIRELESS_ALLOW_TX_RX_AGGR_SET_TID must be set using qapi_WLAN_Set_Param() before calling this API. Similarly, if the user wants to use UAPSD, __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_ENABLE_UAPSD (in SoftAP mode) and __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_UAPSD in non-AP Station mode should be set using qapi_WLAN_Set_Param().

9.1.5.6 qapi_Status_t qapi_WLAN_Set_Callback (qapi_WLAN_Callback_t *callback,* void * *application_Context*)

Sets a callback function and application context (const void *), if any, for WLAN control commands.

Users can use the callback function to handle events received from the WLAN firmware for asynchronous commands. The application context, while setting the callback, will be returned when the callback handler function is called for events. The callback handling is done in the Wi-Fi driver thread's context, hence the event handling should be as fast as possible to avoid performance delays.

Associated data types

qapi_WLAN_Callback_t

Parameters

in	*callback*	Callback function. NULL to clear callback, non-NULL to initialize callback, not support replace.
in	*application_Context*	Application context set by the application.

Returns

QAPI_OK – Application callback handler function for WLAN commands set.
Nonzero value – Setting callback function failed.

Dependencies

The callback function to be set must be a valid function.

9.1.5.7 qapi_Status_t qapi_WLAN_Disconnect (uint8_t *device_ID*)

Disconnects a device or aborts the current connection process. This API internally calls qapi_WLAN_Commit(), hence no explicit call is needed in the disconnect process when using this API.

Parameters

in	*device_ID*	Device ID.

Returns

QAPI_OK – Disconnect process succeeded.
Nonzero value – Disconnection failed.

Dependencies

None.

9.1.5.8 qapi_Status_t qapi_WLAN_Set_Param (uint8_t *device_ID,* uint16_t *group_ID,* uint16_t *param_ID,* const void * *data,* uint32_t *length,* qapi_WLAN_Wait_For_Status_e *wait_For_Status*)

Sets requested parameters in the WLAN firmware.

The group_ID parameter is used to determine to which group the the command belongs, and the param_ID is used to provide the requested command (/parameter).

The __QAPI_WLAN_PARAM_GROUP_* types give more information regarding the possible values for group_ID and param_ID. Currently, only nonblocking calls are supported, hence wait_For_Status should be set to FALSE (0).

Associated data types

qapi_WLAN_Wait_For_Status_e

Parameters

in	*device_ID*	Device ID.
in	*group_ID*	Group ID: system, wireless, security, P2P.
in	*param_ID*	Parameter/command ID.
in	*data*	Value to be set for the requested parameter.
in	*length*	Size of the value to be set.
in	*wait_For_Status*	0: No wait; 1: Wait for result of the set request.

Returns

QAPI_OK – Requested parameter was set.
Nonzero value – Requested parameter was not set.

Dependencies

None.

9.1.5.9 qapi_Status_t qapi_WLAN_Get_Param (uint8_t *device_ID,* uint16_t *group_ID,* uint16_t *param_ID,* void * *data,* uint32_t * *length*)

Retrieves the requested parameter value from the WLAN firmware.

The group_ID parameter is used to determine to which group the the command belongs, and the param_ID is used to provide the requested command (/parameter).

The __QAPI_WLAN_PARAM_GROUP_* types give more information regarding the possible values for group_ID and param_ID.

Parameters

in	*device_ID*	Device ID.
in	*group_ID*	Group ID: system, wireless, security, P2P.
in	*param_ID*	Parameter ID in the given group.
out	*data*	Value retrieved from the WLAN firmware.
out	*length*	Size of the parameter value.

Returns

QAPI_OK – Requested was parameter retrieved from the WLAN firmware.
Nonzero value – Parameter retrieval failed.

Dependencies

None.

9.1.5.10 qapi_Status_t qapi_WLAN_Set_11d (uint32_t *enable*)

Enables or disables 802.11d feature.

If enabled, the country code of the device will be set according to scan results.

Parameters

in	*enable*	- 0: Disable 802.11d. • 1: Enable 802.11d.

Returns

QAPI_OK – Command send success.
Nonzero value – Command failed.

Dependencies

The device works in STA mode, or has not connected with AP.

9.1.5.11 qapi_Status_t qapi_WLAN_Get_11d (uint32_t * *result*)

Determines whether 802.11d feature is enabled or not.

Parameters

out	*result*	Result retrieved from WLAN firmware. • 0: 802.11d disabled. – 1: 802.11d enabled.

Returns

QAPI_OK – Result was retrieved from WLAN firmware.
Nonzero value – Result retrieval failed.

Dependencies

None.

9.1.5.12 qapi_Status_t qapi_WLAN_Get_Country_Code (char * *country_code*)

Gets country code of the device.

Parameters

out	*country_code*	Country code retrieved from the WLAN firmware.

Returns

QAPI_OK – Country code was retrieved from WLAN firmware.
Nonzero value – Country code retrieval failed.

Dependencies

None.

9.1.5.13 qapi_Status_t qapi_WLAN_Get_Regulatory_Info (qapi_WLAN_Reg_Evt_t * *reg*)

Gets regulatory infrmationo of the device.

Parameters

out	*reg*	regulatory information retrieved from the WLAN firmware.

Returns

QAPI_OK – Regulatory information was retrieved from WLAN firmware.
Nonzero value – Regulatory information retrieval failed.

Dependencies

None.

9.1.5.14 qapi_Status_t qapi_WLAN_Set_Rate (qapi_WLAN_Set_Rate_Params_t * *prate_para*)

Set WLAN TX rate.

Parameters

in	*prate_para*	rate config.

Returns

QAPI_OK – Set rate successfully.
Nonzero value – Set rate failed.

Dependencies

None.

9.1.5.15 qapi_Status_t qapi_WLAN_Get_Rate (qapi_WLAN_Set_Rate_Params_t * *prate_para*)

Get WLAN TX rate.

Parameters

in	*prate_para*	rate config.

Returns

QAPI_OK – Set rate successfully.
Nonzero value – Set rate failed.

Dependencies

None.

9.1.5.16 qapi_Status_t qapi_WLAN_Raw_Send (qapi_WLAN_Raw_Send_Params_t * *raw_para*)

Sends raw frame.

Parameters

in	*device_ID*	raw_Params config.

Returns

QAPI_OK – Send frame successfully.

Nonzero value – Send frame failed.

Dependencies

None.

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

Enables the 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.

9.1.5.18 qapi_Status_t qapi_WLAN_Disable_Mgmt_Filter (uint8_t *device_ID*)

Disables the WLAN management frame filter.

Parameters

in	*device_id*	Device ID.

Returns

qapi_Status_t QAPI_OK on success, other error code on failure.

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

Recieves the WLAN management frames which are in the 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.

9.2 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.

9.3 WLAN control operations

9.3.1 Define Documentation

9.3.1.1 #define __QAPI_WLAN_DEVICE0_NAME "wlan0"

Name of first WLAN virtual device, also known as Device 0.

9.3.1.2 #define __QAPI_WLAN_DEVICE1_NAME "wlan1"

Name of second WLAN virtual device, also known as Device 1; currently the maximum number of supported virtual devices is 2 (Device 0 and Device 1).

9.3.1.3 #define __QAPI_WLAN_MAC_LEN 6

Size of the WLAN MAC address in bytes.

9.3.1.4 #define __QAPI_WLAN_MAX_SSID_LEN 32

Maximum size of the WLAN Service Set Identifier (SSID) in bytes.

9.3.1.5 #define __QAPI_WLAN_MAX_SSID_LENGTH (32+1)

Maximum SSID length (includes ending NULL character) in bytes.

9.3.1.6 #define __QAPI_WLAN_PASSPHRASE_LEN 64

Maximum passphrase length for WPA/WPA2 WLAN networks in bytes.

9.3.1.7 #define __QAPI_WLAN_PNO_MAX_NETWORK_PROFILES 5

Maximum number of profiles supported for preferred network offload operation.

9.3.1.8 #define __QAPI_WLAN_IPV6_ADDR_LEN 16

Size of an IPv6 address in bytes.

9.3.1.9 #define __QAPI_WLAN_IPV4_ADDR_LEN 4

Size of an IPv4 address in bytes.

9.3.1.10 #define __QAPI_WLAN_WPS_PIN_LEN 9

Maximum length of a Wi-Fi Protected Setup (WPS) PIN. This length includes the terminating NULL character.

9.3.1.11 #define __QAPI_WLAN_WPS_MAX_KEY_LEN 64

Maximum size of a WPS key in bytes.

9.3.1.12 #define __QAPI_WLAN_PROMISC_MAX_FILTER_IDX 3

Maximum number of filters allowed in Promiscuous mode.

9.3.1.13 #define __QAPI_WLAN_MAX_NUM_CUR_REGDOAMIN_CHANLIST_CHNNELS 64

Maximum number of supported channels that can be retrieved from tbe current regulatory domain channel list.

9.3.1.14 #define __QAPI_WLAN_PROM_FILTER_SOURCE 0x01

Allows the application to enable/disable a source MAC address based filter when operating in Promiscuous mode.

This macro should be set in the filter_flags field of the qapi_WLAN_Promiscuous_Mode_Info_t data structure when issuing qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE.

If this bit is enabled, the application should also set the src_Mac address in the corresponding filter index of qapi_WLAN_Promiscuous_Mode_Info_t.

See also

qapi_WLAN_Promiscuous_Mode_Info_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE
qapi_WLAN_Set_Param

9.3.1.15 #define __QAPI_WLAN_PROM_FILTER_DEST 0x02

Allows the application to enable/disable the destination MAC address based filter when operating in Promiscuous mode.

This macro should be set in the filter_flags field of the qapi_WLAN_Promiscuous_Mode_Info_t data structure when issuing qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE.

If this bit is enabled, the application should also set the dst_Mac address in the corresponding filter index of qapi_WLAN_Promiscuous_Mode_Info_t.

See also

qapi_WLAN_Promiscuous_Mode_Info_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE
qapi_WLAN_Set_Param

9.3.1.16 #define __QAPI_WLAN_PROM_FILTER_FRAME_TYPE 0x04

Allows the application to enable/disable the 802.11 packet type based filter when operating in Promiscuous mode.

This macro should be set in the filter_flags field of the qapi_WLAN_Promiscuous_Mode_Info_t data structure when issuing qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE.

The 802.11 packet type to be matched should be provided in the promisc_frametype field in the corresponding filter index of qapi_WLAN_Promiscuous_Mode_Info_t.

See also

qapi_WLAN_Promiscuous_Mode_Info_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE
qapi_WLAN_Set_Param

9.3.1.17 #define __QAPI_WLAN_PROM_FILTER_FRAME_SUB_TYPE 0x08

Allows the application to enable/disable the 802.11 packet subtype based filter when operating in Promiscuous mode.

This macro should be set in the filter_flags field of the qapi_WLAN_Promiscuous_Mode_Info_t data structure when issuing qapi_WLAN_Set_Param() with __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE.

The 802.11 packet type to be matched should be provided in the promisc_subtype field in the corresponding filter index of qapi_WLAN_Promiscuous_Mode_Info_t.

See also

qapi_WLAN_Promiscuous_Mode_Info_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE
qapi_WLAN_Set_Param

9.3.1.18 #define __QAPI_WLAN_PROMISC_REPORT_RSSI_INFO 0x4000

This macro indicates that the data is RSSI information for a frame in promiscuous mode. When the driver layer reports RSSI information for a frame, this macro is set and the application should treat the data as RSSI information, not a frame context.

9.3.1.19 #define __QAPI_WLAN_PROMISC_INDICATE_APPEND_PAYLOAD 0x8000

This macro indicates that the data is only payload of frame which which has no 802.11 mac header. For AMSDU packet, it has only one wlan 802.11 mac header and may occupy multiple netbufs so the driver layer may report several packets to application layer. If this macro is set, it means the data is only payload and should append to the previous frame.

9.3.1.20 #define __QAPI_WLAN_DBGLOG_MAX_MODULE_ID 64

Total number of Dblog modules supported in the target.

9.3.1.21 #define __QAPI_WLAN_DBGLOG_LOGLEVEL_INFO_LEN 8

Number of word length buffers required to store the DBGLOG Loglevel information for all 64 (maximum supported) modules.

9.3.1.22 #define __QAPI_WLAN_DBGLOG_DEFAULT_MODULE_MASK 0xFFFFFFFFFFFFFFFFLL

Module mask that can be used to enable all modules with a desired loglevel.

9.3.1.23 #define __QAPI_WLAN_DBGLOG_GLOBAL_MODULE_ID 0xFF

Global module ID to set the same configuration(loglevel) for all modulues.

9.3.1.24 #define __QAPI_WLAN_DBGLOG_DEFAULT_MODULE_LOG_LEVEL 0x88888888

Default value to capture only the ERROR loglevel for all 64 modules when dbglog is enabled.

9.3.1.25 #define __QAPI_WLAN_DBGLOG_LOCAL_DBG_PORT 0

Used to configure to send the dbglogs on the local UART on the target chip (Kingfisher).

9.3.1.26 #define __QAPI_WLAN_DBGLOG_REMOTE_DBG_PORT 1

Used to configure to send the dbglogs from the target to the host, which in turn sends them to its UART, where an external tool captures and parses the dbglog binary logs.

9.3.1.27 #define __QAPI_WLAN_DBGLOG_CFG_LOG_LEVEL_FLAG_MASK 0x01

Flag bit to be used to notify the target when a change in loglevel is needed during a dbglog configuration update.

9.3.1.28 #define __QAPI_WLAN_DBGLOG_CFG_DEBUG_PORT_FLAG_MASK 0x02

Flag bit to be used to notify the target when a change in debug port is needed to use during a dbglog configuration update.

9.3.1.29 #define __QAPI_WLAN_DBGLOG_CFG_REPORT_FLAG_MASK 0x04

Flag bit to be used to notify the target when a change in reporting is required or not during a dbglog configuration update.

9.3.1.30 #define __QAPI_WLAN_DBGLOG_CFG_REPORT_SIZE_FLAG_MASK 0x08

Flag bit to be used to notify the target when a change in report size is needed during a dbglog configuration update.

9.3.1.31 #define __QAPI_WLAN_DBGLOG_CFG_TIMER_RESOLUTION_FLAG_MASK 0x10

Flag bit to be used to notify the target when a change in time resolution is needed during a dbglog configuration update.

9.3.1.32 #define __QAPI_WLAN_DBGLOG_CFG_FLUSH_FLAG_MASK 0x20

Flag bit to be used to notify the target to flush log buffer.

9.3.1.33 #define __QAPI_WLAN_DBGLOG_NIBBLE_CNT_IN_WORD_MEMORY 8

Value for the number of nibbles in a word. Used for dbglog loglevel update calculation.

9.3.1.34 #define __QAPI_WLAN_DBGLOG_BIT_CNT_IN_WORD_MEMORY 32

Value for the number of bits in a word memory. Used for dbglog loglevel update calculation.

9.3.1.35 #define __QAPI_WLAN_CHAN_FREQ_1 (2412)

IEEE 802.11 channel 1 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.36 #define __QAPI_WLAN_CHAN_FREQ_2 (2417)

IEEE 802.11 channel 2 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.37 #define __QAPI_WLAN_CHAN_FREQ_3 (2422)

IEEE 802.11 channel 3 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.38 #define __QAPI_WLAN_CHAN_FREQ_4 (2427)

IEEE 802.11 channel 4 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.39 #define __QAPI_WLAN_CHAN_FREQ_5 (2432)

IEEE 802.11 channel 5 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.40 #define __QAPI_WLAN_CHAN_FREQ_6 (2437)

IEEE 802.11 channel 6 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.41 #define __QAPI_WLAN_CHAN_FREQ_7 (2442)

IEEE 802.11 channel 7 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.42 #define __QAPI_WLAN_CHAN_FREQ_8 (2447)

IEEE 802.11 channel 8 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.43 #define __QAPI_WLAN_CHAN_FREQ_9 (2452)

IEEE 802.11 channel 9 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.44 #define __QAPI_WLAN_CHAN_FREQ_10 (2457)

IEEE 802.11 channel 10 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.45 #define __QAPI_WLAN_CHAN_FREQ_11 (2462)

IEEE 802.11 channel 11 in MHz. This channel is allowed in USA, Canada, Europe, and Japan.

9.3.1.46 #define __QAPI_WLAN_CHAN_FREQ_12 (2467)

IEEE 802.11 channel 12 in MHz. This channel is allowed in Europe and Japan.

9.3.1.47 #define __QAPI_WLAN_CHAN_FREQ_13 (2472)

IEEE 802.11 channel 13 in MHz. This channel is allowed in Europe and Japan.

9.3.1.48 #define __QAPI_WLAN_CHAN_FREQ_14 (2484)

IEEE 802.11 channel 14 in MHz. This channel is allowed in Japan.

9.3.1.49 #define __QAPI_WLAN_CHAN_FREQ_36 (5180)

IEEE 802.11a channel 36 in MHz.

9.3.1.50 #define __QAPI_WLAN_CHAN_FREQ_40 (5200)

IEEE 802.11a channel 40 in MHz.

9.3.1.51 #define __QAPI_WLAN_CHAN_FREQ_44 (5220)

IEEE 802.11a channel 44 in MHz.

9.3.1.52 #define __QAPI_WLAN_CHAN_FREQ_48 (5240)

IEEE 802.11a channel 48 in MHz.

9.3.1.53 #define __QAPI_WLAN_CHAN_FREQ_52 (5260)

IEEE 802.11a channel 52 in MHz.

9.3.1.54 #define __QAPI_WLAN_CHAN_FREQ_56 (5280)

IEEE 802.11a channel 56 in MHz.

9.3.1.55 #define __QAPI_WLAN_CHAN_FREQ_60 (5300)

IEEE 802.11a channel 60 in MHz.

9.3.1.56 #define __QAPI_WLAN_CHAN_FREQ_64 (5320)

IEEE 802.11a channel 64 in MHz.

9.3.1.57 #define __QAPI_WLAN_CHAN_FREQ_100 (5500)

IEEE 802.11a channel 100 in MHz.

9.3.1.58 #define __QAPI_WLAN_CHAN_FREQ_104 (5520)

IEEE 802.11a channel 104 in MHz.

9.3.1.59 #define __QAPI_WLAN_CHAN_FREQ_108 (5540)

IEEE 802.11a channel 108 in MHz.

9.3.1.60 #define __QAPI_WLAN_CHAN_FREQ_112 (5560)

IEEE 802.11a channel 112 in MHz.

9.3.1.61 #define __QAPI_WLAN_CHAN_FREQ_116 (5580)

IEEE 802.11a channel 116 in MHz.

9.3.1.62 #define __QAPI_WLAN_CHAN_FREQ_132 (5660)

IEEE 802.11a channel 132 in MHz.

9.3.1.63 #define __QAPI_WLAN_CHAN_FREQ_136 (5680)

IEEE 802.11a channel 136 in MHz.

9.3.1.64 #define __QAPI_WLAN_CHAN_FREQ_140 (5700)

IEEE 802.11a channel 140 in MHz.

9.3.1.65 #define __QAPI_WLAN_CHAN_FREQ_149 (5745)

IEEE 802.11a channel 149 in MHz.

9.3.1.66 #define __QAPI_WLAN_CHAN_FREQ_153 (5765)

IEEE 802.11a channel 153 in MHz.

9.3.1.67 #define __QAPI_WLAN_CHAN_FREQ_157 (5785)

IEEE 802.11a channel 157 in MHz.

9.3.1.68 #define __QAPI_WLAN_CHAN_FREQ_161 (5805)

IEEE 802.11a channel 161 in MHz.

9.3.1.69 #define __QAPI_WLAN_CHAN_FREQ_165 (5825)

IEEE 802.11a channel 165 in MHz.

9.3.1.70 #define __QAPI_WLAN_6G_CHAN_FREQ_1 (5955)

6G channel 1 in MHz.

9.3.1.71 #define __QAPI_WLAN_SECURITY_AUTH_PSK 0x01

Flag that indicates whether a scanned access point's authentication type is of type Pre-Shared Key. This is indicated in the wpa_auth field (for WPA APs) and rsn_Auth field (for WPA2 APs) of the qapi_WLAN_BSS_Scan_Info_t structure.

9.3.1.72 #define __QAPI_WLAN_SECURITY_AUTH_1X 0x02

Flag that indicates whether the scanned access point's authentication type is of type 802.1x(EAP based). This is indicated in the wpa_auth field (for WPA APs) and rsn_Auth field (for WPA2 APs) of the qapi_WLAN_BSS_Scan_Info_t structure.

9.3.1.73 #define __QAPI_WLAN_SECURITY_AUTH_SAE 0x04

Flag that indicates whether the scanned access point's authentication type is of type SAE. This is indicated in the wpa_auth field (for WPA APs) and the rsn_auth field (for WPA2/WPA3 APs) of the qapi_WLAN_BSS_Scan_Info_t structure.

9.3.1.74 #define __QAPI_WLAN_CIPHER_TYPE_TKIP 0x04

Flag that indicates whether the scanned access point's encryption type is of type TKIP (Temporal Key Integrity Protocol). This is indicated in the wpa_Cipher field (for WPA APs) and rsn_Cipher field (for WPA2 APs) of the qapi_WLAN_BSS_Scan_Info_t structure as part of the scan result indication.

9.3.1.75 #define __QAPI_WLAN_CIPHER_TYPE_CCMP 0x08

Flag that indicates whether the scanned access point's encryption type is of type CCMP. This is indicated in the wpa_Cipher field (for WPA APs) and rsn_Cipher field (for WPA2 APs) of the qapi_WLAN_BSS_Scan_Info_t structure as part of the scan result indication.

9.3.1.76 #define __QAPI_WLAN_CIPHER_TYPE_WEP 0x02

Flag that indicates whether the scanned access point's encryption type is of type WEP (Wired Equivalent Privacy). This is indicated in the wpa_Cipher field and rsn_Cipher field of the qapi_WLAN_BSS_Scan_Info_t structure as part of the scan result indication.

9.3.1.77 #define __QAPI_WLAN_MAX_WEP_KEY_SZ 16

Maximum size of the WEP key.

9.3.1.78 #define __QAPI_MAX_SCAN_RESULT_ENTRY 40

Maximum number of scan results that the driver buffer can hold when using QAPI_WLAN_BUFFER_SCAN_RESULTS_BLOCKING_E and QAPI_WLAN_BUFFER_SCAN_RESULTS_NON_BLOCKING_E options to store WLAN scan results.

9.3.1.79 #define __QAPI_WLAN_SHORTSCANRATIO_DEFAULT 3

Macro to indicate the default Short Scan ratio.

9.3.1.80 #define __QAPI_WLAN_MAX_NS_OFFLOAD_TUPLES 2

Macro that indicates the maximum number of network solicitation entries that the WLAN FW can offload. Every tuple here corresponds to one virtual WLAN device.

9.3.1.81 #define __QAPI_WLAN_MAX_ARP_OFFLOAD_TUPLES 2

Macro that indicates the maximum number of ARP entries that the WLAN FW can offload. Every tuple corresponds to one virtual WLAN device.

9.3.1.82 #define __QAPI_WLAN_NSOFF_MAX_TARGET_IPS 2

Macro that indicates the maximum number of network solicitation addresses that the WLAN FW supports per device (tuple); one for a global IPv6 address and another for a link local IPv6 address (interchangable).

9.3.1.83 #define __QAPI_WLAN_START_SCAN_PARAMS_CHANNEL_LIST_MAX 12

Macro that indicates the maximum number of channels that can be issued in a channel list while scanning.

9.3.1.84 #define __QAPI_WLAN_MAX_NUM_FILTERED_EVENTS 64

Macro that indicates the maximum number of WLAN firmware events that can be filtered within the firmware.

9.3.1.85 #define __QAPI_WLAN_VERSION_SUBSTRING_LEN 20

Macro that defines the maximum length of a string for each of the subversion information strings.

9.3.1.86 #define __QAPI_WLAN_PATTERN_MAX_SIZE 128

Macro that indicates the maximum size of patterns supported for packet filtering and WOW filtering features.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.87 #define __QAPI_WLAN_PATTERN_MASK (__QAPI_WLAN_PATTERN_MAX_SIZE/8)

Macro that indicates the maximum size of a supported pattern mask.

This is the function of __QAPI_WLAN_PATTERN_MAX_SIZE as a pattern mask, which is a bitmap that indicates the validity of pattern data where every bit in the pattern mask corresponds to a byte in the pattern data.

For an application, if a pattern to be matched is not a contiguous byte stream, the application should set valid bytes to be matched by setting appropriate bits in the pattern mask member of qapi_WLAN_Add_Pattern_t.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.88 #define __QAPI_WLAN_PATTERN_REJECT_FLAG 0x1

Macro to indicate a filter action mask as a REJECT on finding a matched pattern during WOW and packet filtering.

If this flag is set in pattern_Action_Flag of qapi_WLAN_Add_Pattern_t for a given pattern, the received packet that matches this pattern is dropped by the firmware.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.89 #define __QAPI_WLAN_PATTERN_ACCEPT_FLAG 0x2

Macro to indicate a filter action as ACCEPT on finding a matched pattern during WOW and packet filtering.

If this flag is set in pattern_Action_Flag of qapi_WLAN_Add_Pattern_t for a given pattern, the received packet that matches this pattern is forwarded to the host by the firmware.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.90 #define __QAPI_WLAN_PATTERN_DEFER_FLAG 0x4

Macro to indicate a filter action mask as DEFER on finding a matched pattern during WOW and packet filtering.

If this flag is set in pattern_Action_Flag of qapi_WLAN_Add_Pattern_t for a given pattern, the received packet that matches this pattern defers to the next header type.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.91 #define __QAPI_WLAN_PATTERN_WOW_FLAG 0x80

Macro to indicate that a given pattern is a WOW pattern.

If pattern_Action_Flag has this bit set and if data is received with the pattern matched, the firmware wakes up the host by asserting an out-of-band GPIO interrupt.

If this bit is not set in the pattern_Action_Flag of the pattern added, the pattern is just considered as a packet filter and an out-of-band interrupt will not be asserted on matching the pattern.

See also

qapi_WLAN_Add_Pattern_t

9.3.1.92 #define __QAPI_WLAN_DEFAULT_FILTER_INDEX 0

Default pattern index for WOW and packet filtering.

9.3.1.93 #define __QAPI_WLAN_MAX_FILTER_INDEX 8

Maximum value of the pattern index for WOW and packet filtering.

9.3.1.94 #define __QAPI_WLAN_DELETE_ALL_FILTER_INDEX 0xFF

Pattern index to delete all filters in the protocol header.

9.3.1.95 #define __QAPI_WLAN_DEFAULT_FILTER_PRIORITY 0

Default pattern prioirty for WOW and packet filtering.

9.3.1.96 #define __QAPI_WLAN_TX_STATUS_IDLE 0x01

Macro that indicates that the Tx queue of the WLAN driver is empty. If the driver returns this on querying the Tx status, the application can safely put the WLAN subsystem in Suspend mode.

9.3.1.97 #define __QAPI_WLAN_TX_STATUS_HOST_PENDING 0x02

Macro that indicates that the WLAN driver Tx queue has one or more frames waiting to be transmitted. If the driver returns this on querying the Tx status, the application should not proceed with the suspend operation.

9.3.1.98 #define __QAPI_WLAN_TX_STATUS_WIFI_PENDING 0x03

Macro that indicates that the WLAN driver has not completed transmission of all the frames, even if the Tx queue is empty. If the driver returns this on querying the Tx status, the application should not proceed with the suspend operation.

9.3.1.99 #define __QAPI_WLAN_AGGRX_BUFFER_SIZE_MAX 16

Macro that indicates the maximum number of buffers that can be aggregated by the WLAN subsystem when transmitting an AMPDU frame.

9.3.1.100 #define __QAPI_WLAN_AGGRX_CFG_INVAL 0xFF

Macro to indicate in the host for the firmware not to consider a particular aggregation parameter.

The firmware considers values other than 0xFF as valid aggregation parameter from the host while configuring aggregation parameters. To avoid using invalid aggregation parameters in the target, the host resets all the aggregation parameters to 0xFF (this macro) before configuring the desired parameters.

9.3.2 Enumeration Type Documentation

9.3.2.1 enum qapi_WLAN_Bit_Rate_t

Enumumeration that identifies WLAN rates.

Enumerator:

QAPI_WLAN_RATE_AUTO_E 1 Mbps.
QAPI_WLAN_RATE_1Mb_E 1 Mbps.
QAPI_WLAN_RATE_2Mb_E 1 Mbps.
QAPI_WLAN_RATE_5_5Mb_E 5 Mbps.
QAPI_WLAN_RATE_11Mb_E 11 Mbps.
QAPI_WLAN_RATE_6Mb_E 6 Mbps.
QAPI_WLAN_RATE_9Mb_E 9 Mbps.
QAPI_WLAN_RATE_12Mb_E 12 Mbps.
QAPI_WLAN_RATE_18Mb_E 18 Mbps.
QAPI_WLAN_RATE_24Mb_E 24 Mbps.
QAPI_WLAN_RATE_36Mb_E 36 Mbps.
QAPI_WLAN_RATE_48Mb_E 48 Mbps.
QAPI_WLAN_RATE_54Mb_E 54 Mbps.
QAPI_WLAN_RATE_MCS_0_20_E MCS rate index 0 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_1_20_E MCS rate index 1 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_2_20_E MCS rate index 2 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_3_20_E MCS rate index 3 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_4_20_E MCS rate index 4 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_5_20_E MCS rate index 5 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_6_20_E MCS rate index 6 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_7_20_E MCS rate index 7 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_8_20_E MCS rate index 8 for 20 Mhz channel.
QAPI_WLAN_RATE_MCS_9_20_E MCS rate index 9 for 20 Mhz channel; not supported at this time.
QAPI_WLAN_RATE_MCS_10_20_E MCS rate index 10 for 20 Mhz channel; not supported at this time.
QAPI_WLAN_RATE_MCS_11_20_E MCS rate index 11 for 20 Mhz channel; not supported at this time.
QAPI_WLAN_RATE_MCS_12_20_E MCS rate index 12 for 20 Mhz channel; not supported at this time.
QAPI_WLAN_RATE_MCS_13_20_E MCS rate index 13 for 20 Mhz channel; not supported at this time.
QAPI_WLAN_RATE_MCS_14_20_E MCS rate index 14 for 20 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_15_20_E MCS rate index 15 for 20 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_0_40_E MCS rate index 0 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_1_40_E MCS rate index 1 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_2_40_E MCS rate index 2 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_3_40_E MCS rate index 3 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_4_40_E MCS rate index 4 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_5_40_E MCS rate index 5 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_6_40_E MCS rate index 6 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_7_40_E MCS rate index 7 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_8_40_E MCS rate index 8 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_9_40_E MCS rate index 9 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_10_40_E MCS rate index 10 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_11_40_E MCS rate index 11 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_12_40_E MCS rate index 12 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_13_40_E MCS rate index 13 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_14_40_E MCS rate index 14 for 40 Mhz channel; not supported at this time.

QAPI_WLAN_RATE_MCS_15_40_E MCS rate index 15 for 40 Mhz channel; not supported at this time.

9.4 WLAN parameter group information

9.4.1 Define Documentation

9.4.1.1 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM 0

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

9.4.1.2 #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.

9.4.1.3 #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.

9.4.1.4 #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.

9.4.1.5 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_ENABLE_SUSPEND_RESUME 0

An application can enable/disable suspend/resume operations of a WLAN subsystem by setting this parameter as TRUE (to enable) or FALSE (to disable).

This setting just enables suspend/resume and does not actually put the WLAN subsystem in suspend mode.

The application should use qapi_WLAN_Suspend_Start() to put the WLAN subsystem in Suspend mode.

By default, this feature is enabled by the WLAN driver.

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

See also

qapi_WLAN_Suspend_Start

9.4.1.6 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_ENABLE 1

Used to enable/disable the target (Kingfisher) debug logging.

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

9.4.1.7 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_CONFIG 2

Used to configure dbglog configuration, such as debug port, report enable, report size, etc.

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

9.4.1.8 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_DBGLOG_MODULE_CONFIG 3

Used to configure dbglog module loglevel configuration for a specific module or all modules.

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

9.4.1.9 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_FIRMWARE_VERSION 7

Command ID to obtain the WLAN firmware version by querying the driver.

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

Parameters

out	*uint32*	Variable of type qapi_WLAN_Firmware_Version_String_t, which will be filled by the driver.

See also

qapi_WLAN_Firmware_Version_String_t

9.4.1.10 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_DRIVER_NETBUF_POOL_SIZE 8

Command ID to control the number of buffers in the WLAN driver buffer pool.

This command also allows the application to configure the number of buffers to be reserved for Rx in the driver.

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

Parameters

in	*pool_config_info*	Application populates the buffer pool configuration values in qapi_WLAN_A_Netbuf_Pool_Config_t and passes them to the driver.

See also

qapi_WLAN_A_Netbuf_Pool_Config_t

Side effects

Changing this value can have an impact on WLAN performance.

9.4.1.11 #define __QAPI_WLAN_PARAM_GROUP_SYSTEM_LAST_ERROR 9

Used to get the last system error.

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

9.4.1.12 #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

9.4.1.13 #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.

9.4.1.14 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SCAN_PARAMS 3

Command ID to set scan parameters to the driver. These parameters should be set before performing a scan operation.

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

Parameters

in	*qapi_WLAN_Scan_Params_t*	Address of the variable holding all customizable scan parameters.

See also

qapi_WLAN_Scan_Params_t

9.4.1.15 #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.

9.4.1.16 #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

9.4.1.17 #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

9.4.1.18 #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

9.4.1.19 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROBE_REQ_FWD_TO_HOST 8

Command ID to enable/disable forwarding of incoming probe request frames when operating in Softap mode.

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

Parameters

in	*uint32_t*	Set to TRUE to enable forwarding, FALSE to disable.

9.4.1.20 #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*	Strucuture populated with required aggregation parameters for Tx and Rx.

See also

qapi_WLAN_Aggregation_Params_t

9.4.1.21 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_ROAMING 10

Command ID to enable/disable roaming. Disabling roaming disables any autonomous scans (like background scans) performed by the firmware.

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

Parameters

in	*uint32_t*	Set to 1 to enable roaming, 3 to disable roaming.

9.4.1.22 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PROMISCUOUS_MODE 11

Command ID to enable/disable Promiscuous mode, which is only supported on virtual device 0.

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

Parameters

in	*qapi_WLAN_-Promiscuous_Mode-_Info_t*	Parameters should be set appropriately while enabling.

See also

qapi_WLAN_Promiscuous_Mode_Info_t

9.4.1.23 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_POWER_MODE_POLICY 13

Command ID to set WLAN power mode policy when entering Power Save mode.

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

Parameters

in	*qapi_WLAN_Power_Policy_Params_t*	Policy parameters populated by the application.

See also

qapi_WLAN_Power_Policy_Params_t

9.4.1.24 #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

9.4.1.25 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_WAKE_ON_WIRELESS 17

Command ID to enable/disable Wake on Wireless (WOW).

This command just initializes the WOW feature and does not perform any filtering unless appropriate wake patterns are set. If this feature is enabled, the firmware uses an out-of-band interrupt to awaken the host in case of a pattern match.

The application should only enable this feature if the bus interrupt cannot be configured as a wakeup interrupt. If the bus interrupt can be a wakeup interrupt, the packet filtering feature should be sufficient to perform necessary filtering functionality.

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

Parameters

in	*uint32_t*	Set to TRUE to enable WOW, FALSE to disable.

9.4.1.26 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ADD_PATTERN 18

Command ID to add a new packet filter/WOW filter pattern.

This command allows the application to add one filter pattern at a time. The pattern index value passed by the application should not exceed 8.

Pattern update is currently not supported.

If a given pattern index must be updated, the application should delete the pattern in that index first and then add a pattern again for the same index.

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

Parameters

in	*qapi_WLAN_Add_-Pattern_t*	Parameter with the necessary pattern information.

See also

qapi_WLAN_Add_Pattern_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_DELETE_PATTERN

9.4.1.27 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_GREEN_T-X 19

Command ID to enable/disable the Green TX feature. This is a power optimization feature where WLAN transmit power is reduced when operating under good link conditions.

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

Parameters

in	*uint32_t*	Set to TRUE to enable Green TX, FALSE to disable.

9.4.1.28 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_LOW_POWE-R_LISTEN 20

Command ID to enable/disable the Low Power Listen (LPL) feature. This is a power optimization feature where WLAN listen is compromised by operating some of the components with reduced power. Under good link conditions, this compromise should not have any impact on packet reception.

NOTE This parameter can only be used with qapi_WLAN_Set_Param().Device ID must use 0.

Parameters

in	*uint32_t*	Set to TRUE to enable LPL, FALSE to disable.

9.4.1.29 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RATE 21

Command ID to set (for TX)/get (for RX) the WLAN rate of packets transmitted over radio.

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

Parameters

in,out	*qapi_WLAN_Bit_-Rate_t*	Rate value to be set

See also

qapi_WLAN_Bit_Rate_t

9.4.1.30 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_REG_DOMAIN 22

Command ID to get the current operating regulatory domain from the driver.

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

Parameters

out	*uint32_t*	Current operating regulatory domain that will be filled by the driver.

9.4.1.31 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_TX_STATUS 24

Command ID to get the driver transmit queue status. The application should ensure that no packets are pending for transmission before putting WLAN in Suspend mode.

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

Parameters

out	*uint32_t*	Variable in which the driver returns the Tx status. The driver populates __QAPI_WLAN_TX_STATUS_IDLE if the driver queue is empty, some other value otherwise.

See also

__QAPI_WLAN_TX_STATUS_IDLE __QAPI_WLAN_TX_STATUS_HOST_PENDING __QAPI_WLAN_TX_STATUS_WIFI_PENDING

9.4.1.32 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STATS 25

Command ID to get various statistics information from the WLAN driver/firmware.

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

Parameters

out	*qapi_WLAN_Get_-Statistics_t*	Variable in which WLAN populates stats information collected from the firmware.

See also

qapi_WLAN_Get_Statistics_t

9.4.1.33 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NETWORK_OFFLOAD_ENABLE 26

Command ID to enable/disable the preferred network offload feature.

This command also enables the application to configure various global parameters for the PNO feature. The application should enable the PNO feature after WLAN disconnects from the AP and disable PNO before connecting to an AP.

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

Parameters

in	*qapi_WLAN_-Preferred_Network_-Offload_Config_t*	Holds global configuration parameters for the PNO feature.

See also

qapi_WLAN_Preferred_Network_Offload_Config_t

9.4.1.34 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_PREFERRED_NETWORK_PROFILE 27

Command ID to set the preferred network offload (PNO) profile for which the firmware performs autonomous scans based on the PNO global configuration provided.

This command allows the user to add one SSID profile to be scanned on the given index.

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

Parameters

in	*qapi_WLAN_-Preferred_Network_-Profile_t*	PNO profile information to scan.

See also

qapi_WLAN_Preferred_Network_Profile_t
__QAPI_WLAN_PNO_MAX_NETWORK_PROFILES

9.4.1.35 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_KEEP_ALIVE_IN-_SEC 31

Command ID to configure the keepalive frame interval period in Station mode.

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

Parameters

in	*uint32_t*	Keepalive interval in seconds.

9.4.1.36 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_LISTEN_INTERV-AL_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).

9.4.1.37 #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.

9.4.1.38 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_RSSI_THRESHOLD 34

Used to set the received signal strength indicator threshold.

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

9.4.1.39 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_OFFLOAD_ENABLE 35

Command ID to configure global parameters of the TCP keepalive (KA) offload feature.

This command also allows application to set global parameters when using the TCP KA offload feature.

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

Parameters

in	*qapi_WLAN_TCP_Offload_Enable_t*	Configuration parameters to be set for the TCP KA feature.

See also

qapi_WLAN_TCP_Offload_Enable_t

9.4.1.40 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_TCP_KEEPALIVE_OFFLOAD_SESSION_CFG 36

Command ID to configure session parameters of the TCP keepalive offload session. This command allows application to configure one session information at a time.

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

Parameters

in	*qapi_WLAN_TCP_Offload_Config_Params_t*	Per-session configuration information to be used in a TCP KA session. The maximum number of TCP KA sessions should not exceed 3.

See also

qapi_WLAN_TCP_Offload_Config_Params_t

9.4.1.41 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_UAPSD 37

Command ID to enable/disable unscheduled automatic power save delivery (UAPSD) in Station mode. This should be done before associating to an access point.

Currently, this command only supports all ACs or none, and explicit UAPSD (ADDTS) is not supported.

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

Parameters

in	*uint32_t*	Bitmask of access categories for which UAPSD is to be enabled. Following are the bitmask mappings for various classes of traffic: • Bit 0 – Best effort traffic • Bit 1 – Background traffic • Bit 2 – Video traffic • Bit 3 – Voice traffic

9.4.1.42 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_STA_MAX_SP_LEN 38

Used to set the maximum number of total buffered MSDUs and MMPDUs delivered by the AP to the station during a service period.

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

9.4.1.43 #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.

9.4.1.44 #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.

9.4.1.45 #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()

9.4.1.46 #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.

9.4.1.47 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_WPS_FLAG 44

Command ID to enable/disable the WPS 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 WPS 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_Commit()
__QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE

9.4.1.48 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_DTIM_INTERVAL 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

9.4.1.49 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ARP_OFFLOAD_PARAMS 48

Command ID to set Address Resolution Protocol (ARP) offload parameters for the given virtual device. This should only be used when operating in Station mode.

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

Parameters

in	*qapi_WLAN_ARP_Offload_Config_t*	ARP offload configuration parameters.

See also

qapi_WLAN_ARP_Offload_Config_t

9.4.1.50 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_NS_OFFLOAD_PARAMS 49

Command ID to set network solicitation (NS) offload parameters for the given virtual device. This should only be used when operating in Station mode.

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

Parameters

in	*qapi_WLAN_NS_Offload_Config_t*	NS offload configuration parameters.

See also

qapi_WLAN_NS_Offload_Config_t

9.4.1.51 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ENABLE_PKT_FILTER 50

Command ID to enable/disable the packet filtering feature.

This command just initializes the packet filtering feature and does not perform any filtering until packet filter patterns are set.

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

Parameters

in	*uint32_t*	Set to TRUE to enable packet filtering, FALSE to disable.

9.4.1.52 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_DELETE_PATTERN 51

Command ID to delete a patten in a given pattern index. The pattern index should not exceed 8.

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

Parameters

in	*uint32_t*	Pattern index to be deleted.
in	*uint32_t*	Pattern header type.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_ADD_PATTERN

9.4.1.53 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_ENABLE_UAPSD 52

Command ID to enable/disable UAPSD in SoftAP mode.

UAPSD must be enabled in SoftAP after setting the operating mode as AP and before committing the AP profile (qapi_WLAN_Commit()) to start SoftAP.

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

Parameters

in	*uint32_t*	Set to TRUE to enable UAPSD, 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_Commit
__QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE

9.4.1.54 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AGGRX_CONFIG 53

Command ID to configure A-MPDU parameters for an AMPDU session.

These parameters should be set before establishing the connection in both STA and AP modes.

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

Parameters

in	*qapi_WLAN_Rx_Aggrx_Params_t*	Aggregation parameters to be set.

See also

qapi_WLAN_Rx_Aggrx_Params_t

9.4.1.55 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG 54

Command ID to enable/disable 802.11v functionality for the WLAN subsystem.

This command instantiates WNM context in the WLAN subsystem but does not actually enable any WNM features.

Applications should use other WNM commands to enable required features and enable the WNM configuration using this command before enabling any WNM features.

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

Parameters

in	*uint8_t*	Set to TRUE to enable WNM, FALSE to disable

9.4.1.56 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_STA_SLEEP_PERIOD 55

Command ID to enter/exit 802.11v WNM sleep along with the sleep parameters. This command is only supported only in Station mode.

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

Parameters

in	*qapi_WLAN_WNM_Sleep_Period_t*	Parameters to enter/exit WNM sleep.

Dependencies

Applications should enable WNM using __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG before enabling the WNM sleep feature.

See also

qapi_WLAN_WNM_Sleep_Period_t
__QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG

9.4.1.57 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_AP_BSS_MAX_IDLE_PERIOD 56

Command ID to enable/disable the WNM BSS maximum idle period feature.

This command is only supported in SoftAP mode.

If this feature is enabled, SoftAP beacons will start including the WNM BSS maximum idle period IE.

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

Parameters

in	*qapi_WLAN_BSS_Max_Idle_Period_t*	Parameter to enable the WNM BSS maximum idle period feature in SoftAP.

Dependencies

Applications should enable WNM using __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG before enabling the WNM BSS maximum idle feature.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_CONFIG
qapi_WLAN_BSS_Max_Idle_Period_t

9.4.1.58 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_CHANNEL_SWITCH 58

Command ID to initiate a channel switch in SoftAP mode.

This command enables SoftAP to send a channel switch request in its beacons to all its associated clients for given periods and changes SoftAP's operating channel to a new channel requested by the application upon completion of a requested period.

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

Parameters

in	*qapi_WLAN_-Channel_Switch_t*	Parameters to be set for a channel switch request.

Dependencies

SoftAP must be up and running.

See also

qapi_WLAN_Channel_Switch_t

9.4.1.59 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EVENT_FILTER 59

Command ID to set up event filters in the WLAN subsystem.

This command enables an applications to choose events that are not in its interest so that the firmware can skip sending those events to hosts, thereby avoiding unnecessary host wakeups.

Note that all the events are not filterable. Event filters should be set for every virtual device. The maximum number of events that can be filtered for each virtual device should not exceed __QAPI_WLAN_MAX_NUM_FILTERED_EVENTS.

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

Parameters

in	*qapi_WLAN_Event_-Filter_t*	Event filter configuration to be set.

See also

qapi_WLAN_Filterable_Event_e
qapi_WLAN_Event_Filter_t
__QAPI_WLAN_MAX_NUM_FILTERED_EVENTS

9.4.1.60 #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 be used with qapi_WLAN_Set_Param() and qapi_WLAN_Get_Param().

Parameters

in,out	*qapi_WLAN_11n_HT_Config_e*	Required 11n HT configuration must be specified.

See also

qapi_WLAN_11n_HT_Config_e

9.4.1.61 #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

9.4.1.62 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_CHANGE_DEFAULT_FILTER_ACTION 62

Command ID to change the action for the default filter for a given header type.

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

Parameters

in	*uint32_t*	Pattern action flag.
in	*uint32_t*	Pattern header type.

See also

__QAPI_WLAN_PARAM_GROUP_WIRELESS_ADD_PATTERN

9.4.1.63 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_COUNTRY_CODE 63

Command ID to configure the country code for AP mode.

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

Parameters

in	*uint8_t[ ]*	Country code string.

9.4.1.64 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_CURR_REGDOMAIN_CHANNEL_LIST 70

Command ID to get the channel list for the current regulatory setting.

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

Parameters

out	*qapi_WLAN_Get_Channel_List_t*	Variable in which WLAN populates channel information collected from the firmware.

See also

qapi_WLAN_Get_Channel_List_t

9.4.1.65 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_ANT_DIV 71

Command ID to set antenna diversity, such as enable or disable antenna diversity, tx antenna follows rx antenna or not, the antenna switch mode, the time interval or the number of packet used for antenna selection.

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

Parameters

in	*qapi_WLAN_Ant_Div_Config_t*	Variable used to set antenna diversity.

See also

qapi_WLAN_Ant_Div_Config_t

9.4.1.66 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_ANT_DIV 72

Command ID to get the status of antenna diversity, such as the current rx physical antenna in 2g or 5g, the current tx physical antenna in 2g or 5g, the average main rssi, the average alternative rssi and so on.

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

Parameters

out	*qapi_WLAN_Get_Ant_Div_t*	Variable in which WLAN populates the status of antenna diversity information collected from the firmware

See also

qapi_WLAN_Get_Ant_Div_t

9.4.1.67 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_ANTENNA 73

Command ID to set physical antenna.

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

Parameters

in	*uint8_t*	the physical antenna number

9.4.1.68 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_DISABLE_CHANNEL 74

Command ID to set/get the authentication 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_Auth_-Mode_e*	Authentication mode to be set.

Dependencies

Authentication mode must be set before connecting to the peer.

See also

qapi_WLAN_Auth_Mode_e Internal use.

9.4.1.69 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EXT_BOARDDATA_INDEX 75

0: will use default bData, 1: will use ext-bData 1, for future, '2' will use ext-bData 2, etc...

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

9.4.1.70 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_GET_STA_DTIM 76

Command ID to get the DTIM value when device is working as station in connection status. If device is not working as station, or device is disconnected, then returned value will be 0.

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

Parameters

out	*uint8_t*	DTIM value. Unit:ms

See also

uint8_t

9.4.1.71 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_AP_WPS_SIMPLE_CONFIGURATION_STATE 77

Command ID to enable/disable the WIFI simple configuration state 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 1 to configure wps state, 0 to be unconfigured.

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_Commit()
__QAPI_WLAN_PARAM_GROUP_WIRELESS_OPERATION_MODE

9.4.1.72 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_ROAM_CONTROL 78

This definition relates to WLAN_LowRssi_Control_Parameters.

Parameters

in	*qapi_WLAN_Low-Rssi_RoamControl_-Params_t*	Set roam low RSSI threshold.

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

9.4.1.73 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_CONNECT_POLICY 79

Command ID to configure connection policy. If device is not working as a station, then returned value will be -1.

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

Parameters

in	*qapi_WLAN_-Connect_Policy_t*	Sets connection policy.

See also

qapi_WLAN_Connect_Policy_t

9.4.1.74 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_SET_BMCAST_RX_FILTER 80

Command ID that sets broadcast and multicast Rx filter (only in XSTICK mode).

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

Parameters

in	*qapi_WLAN_Set_B-Mcast_Filter_t*	Enables broadcast and multicast Rx filter.

See also

qapi_WLAN_Set_BMcast_Filter_t

9.4.1.75 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_WNM_STA_GET_BSS_-MAX_IDLE_PERIOD 81

Command ID that gets the WNM BSS maximum idle period of AP.

NOTE This command is only supported in Station mode.

Parameters

out	*uint16_t*	BSS maximum period value. Idle timeout = 1000TU ∗ BSS Max Idle Period (ms).

See also

uint16_t

9.4.1.76 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_CONCURRENCY_MOD-E 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	*qapi_WLAN_DEV_-Mode_e*	Concurrency mode address

See also

qapi_WLAN_DEV_Mode_e

9.4.1.77 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_MGMT_FRAME_FILTE-R 83

Command ID to set/get the management frames filter to send to the application in the WLAN subsystem.

NOTE This parameter can be used with qapi_WLAN_Enable_Mgmt_Filter() and qapi_WLAN_Disable_Mgmt_Filter().

Parameters

in,out	*qapi_WLAN_MGM-T_FRAME_e*	Management frame type

See also

qapi_WLAN_MGMT_FRAME_e

9.4.1.78 #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 only be used with qapi_WLAN_Set_Param().

Parameters

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

9.4.1.79 #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 only be used with qapi_WLAN_Set_Param().

Parameters

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

9.4.1.80 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCA_PARAM 86

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

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

Parameters

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

See also

qapi_WLAN_Edca_Params_t

9.4.1.81 #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 only be used with qapi_WLAN_Set_Param().

Parameters

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

9.4.1.82 #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 only be used with qapi_WLAN_Set_Param().

Parameters

in	*qapi_WLAN_BA_-Window_Params_t*	Set BA window size.

See also

qapi_WLAN_BA_Window_Params_t

9.4.1.83 #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 only be used with qapi_WLAN_Set_Param().

Parameters

in	*uint32_t*	Change slot time to 9 us or 20 us.

9.4.1.84 #define __QAPI_WLAN_PARAM_GROUP_WIRELESS_EDCCA_THRESHOL-D 90

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

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

Set EDCCA threshold to 0-100.

9.4.1.85 #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

9.4.1.86 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PMK 2

Command ID to set the pairwise master key for the upcoming WPA/WPA2 association procedure.

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

Parameters

in	*uint8_t[ ]*	Pairwise master key to be set. The master key should be of length __QAPI_WLAN_PASSPHRASE_LEN.

Dependencies

This should be done before initiating an association.

See also

__QAPI_WLAN_PASSPHRASE_LEN

9.4.1.87 #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

9.4.1.88 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WEP_KEY_INDEX 4

Command ID to set the active WEP key index for an upcoming association.

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

Parameters

in,out	*uint32_t*	Active key index to be set. The value should be set so that 1 <= index <= 4.

Dependencies

This should be done before initiating an association.

9.4.1.89 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WEP_KEY_PAIR 5

Command ID to set a WEP {key index , key} pair for an upcoming association.

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

Parameters

in,out	*qapi_WLAN_-Security_Wep_Key_-Pair_Params_t*	Parameters to set the WEP key parameters.

Dependencies

This should be done before initiating an association.

9.4.1.90 #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

9.4.1.91 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_METHOD 7

Command ID to set/get the 802.1x method.

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

Parameters

in,out	*qapi_WLAN_8021x-_Method_e*	802.1x mode to be set.

See also

qapi_WLAN_8021x_Method_e

9.4.1.92 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_START (__QAPI-_WLAN_PARAM_GROUP_SECURITY_8021X_METHOD)

The first Command ID to set/get the 802.1x information.

9.4.1.93 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_START (__QAPI-_WLAN_PARAM_GROUP_SECURITY_8021X_START)

The first Command ID to set/get wlan_lib information.

9.4.1.94 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_IDENTITY 8

Set the 802.1x identity for PEAP/TTLS method.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_IDENTITY qapi_WLAN_Set_Param

9.4.1.95 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_USERNAME 9

Set the 802.1x username for PEAP/TTLS method.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_USERNAME qapi_WLAN_Set_Param

9.4.1.96 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PASSWORD 10

Set the 802.1x password for PEAP/TTLS method.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PASSWORD qapi_WLAN_Set_Param

9.4.1.97 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CA_CER 11

Set/get the 802.1x CA certificate filename.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CA_CERT qapi_WLAN_Set_Param qapi_WLAN_Get_Param

9.4.1.98 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CER 12

Set/get the 802.1x certificate filename.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_CERT qapi_WLAN_Set_Param qapi_WLAN_Get_Param

9.4.1.99 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PRIVATE_KEY 13

Set 802.1x private key filename and its password.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_PRIVATE_KEY qapi_WLAN_Set_Param

9.4.1.100 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_NO_SERVER_AUTH 14

Set server authentication override flag.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_NO_SERVER_AUTH qapi_WLAN_Set_Param

9.4.1.101 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_END 30

The last Command ID to set/get 802.1x information.

9.4.1.102 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_MAIN_START (__QAPI_WLAN_PARAM_GROUP_SECURITY_8021X_END+1)

The first Command ID to set/get WLAN library information.

9.4.1.103 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_DEBUG_LEVEL 31

Set security debug level.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_DEBUG_LEVEL qapi_WLAN_Set_Param

9.4.1.104 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PRIV 32

Set security priv.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_PRIV qapi_WLAN_Set_Param

9.4.1.105 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_MAIN_END (60)

The last Command ID to set/get the WLAN library information.

9.4.1.106 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SUPPL_MAIN_START (__QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_MAIN_END+1)

The first Command ID to set/get the supplicant main information.

9.4.1.107 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SUPPL_MAIN_END (90)

The last Command ID to set/get the supplicant main information.

9.4.1.108 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_START (__QAPI_WLAN_PARAM_GROUP_SECURITY_SUPPL_MAIN_END+1)

The first Command ID to set/get the PMF information.

9.4.1.109 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SET_RSN_CAP (__QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_START)

Set Protected Management Frame (PMF) capability.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_SET_RSN_CAP
qapi_WLAN_Set_Param

9.4.1.110 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_END (120)

The last Command ID to set/get PMF information.

9.4.1.111 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_START (__QAPI_WLAN_PARAM_GROUP_SECURITY_PMF_END+1)

The first Command ID to set/get Simultaneous Authentication of Equals (SAE) information.

9.4.1.112 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_FINITE_CYCLIC_GROUP (__QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_START)

Set finite cyclic group for SAE.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_FINITE_CYCLIC_GROUP
qapi_WLAN_Set_Param

9.4.1.113 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_PMK_CACHE (122)

Set PMK cache for sae.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_PMK_CACHE qapi_WLAN_Set_Param

9.4.1.114 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_SAE_END (150)

The last Command ID to set/get SAE information.

9.4.1.115 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_WLIB_END 1000

The last Command ID to set/get wlan_lib information.

9.4.1.116 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_PMKID 1001

Set PMKID.

See also

__QAPI_WLAN_PARAM_GROUP_SECURITY_PMKID qapi_WLAN_Set_Param

9.4.1.117 #define __QAPI_WLAN_PARAM_GROUP_SECURITY_GET_RSN_CAP 1002

Command ID that gets the Robust Security Network (RSN) capability.

NOTE This command is only supported in Station mode.

Parameters

out	*uint16_t*	

See also

uint16_t

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.
- 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.

19.6 Types

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 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 product 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.