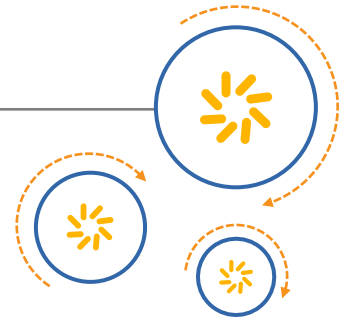




Qualcomm Technologies, Inc.



QCA4010/QCA4012 Hostless SDK (QCA4010.TX.2.0)

User Guide

80-YA116-14 Rev. A

March 15, 2017

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

© 2017 Qualcomm Technologies, Inc. All rights reserved.

Qualcomm Technologies, Inc. (“QTI”) and its affiliates reserve the right to make any updates, corrections and any other modifications to its documentation. The information provided in this document represents QTI’s knowledge and belief as of the date this document is provided. QTI makes no representation or warranty as to the accuracy of such information, and QTI assumes no liability for any use of the information in this documentation. You should obtain the latest information before placing orders for any hardware, and you should verify that such information is current and complete. Information published by QTI regarding any third-party products does not constitute a license to use such products or a warranty or endorsement thereof. Use of such information may require a license from a third party under the intellectual property rights of such third party, or a license from QTI or its affiliates under the intellectual property rights of QTI or its affiliates.

All hardware, equipment, components or products are sold subject to QTI’s (or such other QTI affiliated company that is designated by QTI) standard terms and conditions of sale, as applicable. Notwithstanding anything to the contrary in this documentation or otherwise: (i) you do not receive any rights, licenses, or immunities from suit under any patents of Qualcomm Incorporated, QTI or their respective affiliates as a result of receiving this documentation (whether expressly, impliedly, by virtue of estoppel or exhaustion, or otherwise), (ii) without limitation, you shall not use or sell any wireless wide area network (“WWAN”) baseband integrated circuit that you purchase or acquire from QTI or any product that incorporates any such WWAN baseband integrated circuit (whether alone or in combination with any other software or components) without a separate license or non-assertion covenant from Qualcomm Incorporated in respect of or under all applicable patents, (iii) nothing in this document modifies or abrogates your obligations under any license or other agreement between you and Qualcomm Incorporated, including without limitation any obligation to pay any royalties, and (iv) you will not contend that you have obtained any right, license, or immunity from suit with respect to any patents of Qualcomm Incorporated, QTI or their respective affiliates under or as a result of receiving this documentation (whether expressly, impliedly, by virtue of estoppel or exhaustion, or otherwise).

Revision history

Revision	Date	Description
A	March 2017	Initial release

Contents

1 Introduction	14
2 Board Setup.....	16
2.1 RB01/RB02 board setup	16
2.1.1 Access UART interface in hostless mode.....	16
2.1.2 Access interfaces in JTAG mode.....	17
2.1.3 Access USB interface in calibration mode	18
3 Hostless SDK Shell.....	19
3.1 Hostless system setup	19
3.2 Hostless SDK shell for throughput and power measurement.....	20
3.2.1 Measuring throughput.....	20
3.2.2 Measuring power	21
3.3 Console command reference	22
3.3.1 help.....	24
3.3.2 version.....	24
3.3.3 connect	24
3.3.4 wepkey	24
3.3.5 passphrase	25
3.3.6 wps	25
3.3.7 disc	26
3.3.8 settxpower	26
3.3.9 pwrmode	26
3.3.10 channel	26
3.3.11 wmode	26
3.3.12 listen	27
3.3.13 rssi	27
3.3.14 pmparams.....	27
3.3.15 scanctrl	28
3.3.16 setscanpara	28
3.3.17 setscan	29
3.3.18 driver.....	29
3.3.19 allow_aggr	29
3.3.20 iwconfig scan	29
3.3.21 reset.....	30
3.3.22 ipconfig	30
3.3.23 ipstatic	30
3.3.24 ipdhcp	30
3.3.25 mode.....	30
3.3.26 ap bconint.....	31
3.3.27 ap setmaxstanum	31
3.3.28 ipdhcppool	31
3.3.29 ipv6rtprfx	31
3.3.30 ip_dns_client.....	31
3.3.31 ip_dns_server	31
3.3.32 ip_dns_server_addr	32
3.3.33 ip_dns_local_domain	32
3.3.34 ip_dns add	32
3.3.35 ip_dns delete	32
3.3.36 ip_dns_delete_server_addr	32
3.3.37 ip_resolve_hostname	32

3.3.38 ip_gETHOSTbyname.....	33
3.3.39 ip_gETHOSTbyname2.....	33
3.3.40 ipbridgemode.....	33
3.3.41 tcp_backoff_retry.....	33
3.3.42 settcptimeout.....	33
3.3.43 setmaintimeout.....	33
3.3.44 ip_snmp_client.....	33
3.3.45 ip_snmp_srvr.....	34
3.3.46 ip_snmp_get_time.....	34
3.3.47 ip_snmp_get_time_of_day.....	34
3.3.48 ip_snmp_zone.....	34
3.3.49 ip_show_snmpconfig.....	34
3.3.50 ip_http_server.....	34
3.3.51 ip_http_set_custom_uri.....	35
3.3.52 ip_http_redirect_unknown_url.....	35
3.3.53 ip_http_server_add_redirected_html.....	35
3.3.54 ip_http_set_redirected_url.....	35
3.3.55 ip_http_restrict_http_request.....	35
3.3.56 ip_http_post.....	35
3.3.57 ip_http_get.....	36
3.3.58 ip_http_client.....	36
3.3.59 ssl_start.....	36
3.3.60 ssl_stop.....	36
3.3.61 ssl_config.....	37
3.3.62 ssl_add_cert.....	37
3.3.63 ssl_store_cert.....	37
3.3.64 ssl_delete_cert.....	37
3.3.65 ssl_list_cert.....	37
3.3.66 wdt.....	38
3.3.67 device.....	38
3.3.68 ota_upgrade.....	38
3.3.69 ota_read.....	38
3.3.70 ota_done.....	39
3.3.71 i2c.....	39
3.3.72 pwm.....	39
3.3.73 i2s.....	40
3.3.74 adc.....	40
3.3.75 uart.....	40
3.3.76 setbaudrate.....	40
3.3.77 pingld.....	40
3.3.78 ip_dns_timeout.....	41
3.3.79 mcastfilter.....	41
3.3.80 print.....	41
3.3.81 ota_ftp.....	41
3.3.82 ota_http.....	42
3.3.83 ota_https.....	42
3.3.84 bmiss.....	43
3.3.85 appleie.....	43
3.3.86 ani.....	43
3.3.87 partition_index.....	43
3.3.88 timer_socket.....	43
3.4 Benchmark component.....	44
3.4.1 Benchmark commands.....	44
3.4.2 Multi-stream test.....	46
3.4.3 Throughput command line tool.....	48
3.5 HTTP Server.....	49
3.5.1 Dynamic HTML.....	49
3.5.2 HTTP server GET and POST method test.....	50
3.5.3 Example HTML file.....	50
3.6 HTTP Client.....	51
3.6.1 HTTP PUT, PATCH, GET and POST methods test.....	51
3.6.2 Setup example.....	52

3.7 Multi-session HTTP Client	52
3.7.1 Multi-session HTTP client test methods on CLI	52
3.7.2 Multi-session HTTP client setup example	53
3.8 Simple DNS Server	54
3.8.1 Simple DNS Server test (Soft AP mode)	54
3.9 DNS Client	55
3.10 OTA firmware upgrade	56
3.10.1 TFTP OTA firmware upgrade methods	56
3.11 Fail-Safe flash partitioning	59
3.12 WPS demo application	59
3.12.1 Station mode	59
3.12.2 Soft AP mode	60
3.13 SSL	61
3.13.1 Certificate management demo	62
3.13.2 Cipher Suites list	64
3.14 Data set	65
3.15 Dynamic GPIO configuration	66
3.15.1 GPIO customization	66
3.15.2 Pin-based active configuration set	69
3.15.3 Adding pin configurations to the GPIO framework	70
3.15.4 GPIO mode of operation	72
3.15.5 Interrupt registration	73
3.15.6 Suspend/resume	73
4 P2P Demo	75
4.1 P2P commands	75
4.1.1 on off	76
4.1.2 status	76
4.1.3 cancel	76
4.1.4 nodelist	76
4.1.5 find	77
4.1.6 list_network	77
4.1.7 autogo	77
4.1.8 auth	77
4.1.9 connect	78
4.1.10 invite	78
4.1.11 invite_auth	78
4.1.12 join	79
4.1.13 prov	79
4.1.14 set	79
4.1.15 setconfig	80
4.1.16 start_pers	80
4.1.17 passphrase	80
4.1.18 setoppps	80
4.1.19 setnoa	80
4.1.20 get	81
4.1.21 listen	81
4.2 P2P demo setup	81
4.2.1 Windows PC1 setup	82
4.2.2 Ubuntu PC2 setup	82
4.3 P2P find and provision	85
4.3.1 Wi-Fi module using PUSH method	85
4.3.2 Development kit using KEYPAD or DISPLAY method	86
4.4 P2P connection with PUSH on the development kit	87
4.4.1 Development kit as P2P Client or P2P GO	87
4.5 P2P connection with KEYPAD on the development kit	90
4.5.1 Development kit as P2P Client or P2P GO	90
4.6 P2P connection with DISPLAY on the development kit	91
4.6.1 Development kit as P2P Client or P2P GO	91

4.7 Autonomous GO on the development kit	92
4.7.1 Peer device joining Auto GO on the development kit	92
4.8 Accept Invitation	94
4.8.1 Development kit accepting invitation from peer device	94
4.9 Re-invoke persistent group	96
4.9.1 Development kit re-invoke peer device	96
4.9.2 Peer device re-invoke the development kit	100
5 Concurrency Demo	104
5.1 Setup	104
6 Power Measurement	106
6.1 Overview	106
6.1.1 Test environment	106
6.1.2 Power test preparations	106
6.1.3 Power measurement points	107
6.1.4 Console connection	108
6.2 Measuring suspend current	108
6.3 Measuring sleep current	108
6.4 Measuring DTIM time-averaged current	109
6.4.1 Measuring DTIM1 current	109
6.4.2 Measuring DTIM3 current	109
6.4.3 Measuring DTIM5 current	110
6.4.4 Measuring DTIM10 current	110
6.5 Measuring MAX Perf connection idle current	111
6.6 Measuring TCP Tx/Rx current	111
6.7 Measuring Green Tx current (UDP uplink)	111
6.8 Measuring LPL current (UDP downlink)	112
7 APIs 113	
7.1 System overview	113
7.2 Hostless SDK	113
7.2.1 Target domain	114
7.2.2 PseudoHost domain	114
7.2.3 System memory map	115
7.3 Network APIs	116
7.3.1 qcom_ipconfig	117
7.3.2 qcom_ip6_address_get	117
7.3.3 qcom_ping	117
7.3.4 qcom_ping6	117
7.3.5 qcom_ip6config_router_prefix	118
7.3.6 qcom_bridge_mode_enable	118
7.3.7 qcom_set_hostname	118
7.3.8 qcom_dhcps_set_pool	118
7.3.9 qcom_dhcps_release_pool	119
7.3.10 qcom_dns_server_address_get	119
7.3.11 qcom_dnss_enable	119
7.3.12 qcom_dns_local_domain	119
7.3.13 qcom_dnsc_enable	120
7.3.14 qcom_dnsc_add_server_address	120
7.3.15 qcom_dnsc_del_server_address	120
7.3.16 qcom_dnsc_get_host_by_name	120
7.3.17 qcom_dns_entry_create	120
7.3.18 qcom_dns_entry_delete	121
7.3.19 qcom_dns_6entry_create	121
7.3.20 qcom_dns_6entry_delete	121
7.3.21 qcom_dnsc_get_host_by_name2	121
7.3.22 qcom_sntp_srvr_addr	122

7.3.23	qcom_sntp_get_time	122
7.3.24	qcom_sntp_get_time_of_day	122
7.3.25	qcom_sntp_zone	122
7.3.26	qcom_sntp_query_srvr_address	123
7.3.27	qcom_enable_sntp_client	123
7.3.28	qcom_tcp_set_exp_backoff	123
7.3.29	qcom_ip4_route	123
7.3.30	qcom_ip6_route	124
7.3.31	qcom_tcp_conn_timeout	124
7.3.32	qcom_socket()	124
7.3.33	qcom_connect()	125
7.3.34	qcom_bind()	125
7.3.35	qcom_listen()	125
7.3.36	qcom_accept()	126
7.3.37	qcom_setsockopt()	126
7.3.38	qcom_getsockopt()	126
7.3.39	qcom_sendto()	127
7.3.40	qcom_send()	127
7.3.41	qcom_recvfrom()	128
7.3.42	qcom_recv()	128
7.3.43	qcom_socket_close()	128
7.3.44	qcom_select()	129
7.3.45	qcom_http_server()	129
7.3.46	qcom_http_server_method()	129
7.3.47	qcom_http_set_post_cb	130
7.3.48	qcom_http_set_get_cb	130
7.3.49	qcom_http_set_custom_uri	130
7.3.50	qcom_http_get_datasend	131
7.3.51	qcom_http_redirect_unknown_url_enable	131
7.3.52	qcom_http_server_add_redirected_page	131
7.3.53	qcom_restrict_http_request	131
7.3.54	qcom_http_set_redirected_url	132
7.3.55	qcom_http_client_method	132
7.3.56	qcom_http_client_body	133
7.3.57	qcom_http_client_register_cb	133
7.3.58	qcom_http_client_connect	133
7.3.59	qcom_http_client_disconnect	134
7.3.60	qcom_http_client_request	134
7.3.61	qcom_http_client_set_body	134
7.3.62	qcom_http_client_set_param	135
7.3.63	qcom_http_client_add_header	135
7.3.64	qcom_http_client_clear_header	135
7.3.65	qcom_ota_upgrade()	135
7.3.66	qcom_ota_done()	136
7.3.67	qcom_ota_session_start	136
7.3.68	qcom_ota_partition_get_size	137
7.3.69	qcom_ota_partition_erase	138
7.3.70	qcom_ota_partition_erase_sectors	138
7.3.71	qcom_ota_parse_image_hdr	138
7.3.72	qcom_ota_partition_verify_checksum	138
7.3.73	qcom_ota_partition_read_data	139
7.3.74	qcom_ota_partition_write_data	139
7.3.75	qcom_ota_session_end	139
7.3.76	qcom_ota_partition_format	140
7.3.77	qcom_read_ota_area	140
7.3.78	qcom_set_ping_id	140
7.3.79	qcom_dhcpc_set_timeout	140
7.3.80	qcom_dhcpc_register_cb	141
7.3.81	qcom_dhcps_register_cb	141
7.4	Wireless Networking APIs	141
7.4.1	qcom_get_scan	141
7.4.2	qcom_set_ssid	142

7.4.3 qcom_get_ssid	142
7.4.4 qcom_set_connect_callback.....	142
7.4.5 qcom_sta_connect_legacy	143
7.4.6 qcom_commit	143
7.4.7 qcom_sta_reconnect_start	143
7.4.8 qcom_set_keep_alive.....	143
7.4.9 qcom_roam_mode_enable	144
7.4.10 qcom_sta_reconnect_stop.....	144
7.4.11 qcom_sta_get_rssi	144
7.4.12 qcom_sta_set_listen_time	144
7.4.13 qcom_sta_get_listen_time	144
7.4.14 qcom_ap_set_beacon_interval.....	145
7.4.15 qcom_ap_get_beacon_interval.....	145
7.4.16 qcom_ap_set_inact_time.....	145
7.4.17 qcom_ap_get_sta_info	145
7.4.18 qcom_buffer_info_get.....	146
7.4.19 qcom_firmware_version_get.....	146
7.4.20 qcom_sec_set_wep_mode.....	146
7.4.21 qcom_sec_get_wep_mode.....	146
7.4.22 qcom_thread_msleep	147
7.4.23 qcom_set_lpl_enable.....	147
7.4.24 qcom_set_gtx_enable.....	147
7.4.25 qcom_set_rate.....	147
7.4.26 qcom_set_scan_timeout.....	147
7.4.27 qcom_set_scan	148
7.4.28 qcom_get_bss_entry_by_ssid	148
7.4.29 qcom_uart_init.....	149
7.4.30 qcom_single_uart_init.....	149
7.4.31 qcom_uart_set_buffer_size	149
7.4.32 qcom_get_uart_config.....	150
7.4.33 qcom_set_uart_config	150
7.4.34 qcom_uart_open.....	150
7.4.35 qcom_uart_close	150
7.4.36 qcom_uart_read	151
7.4.37 qcom_uart_write	151
7.4.38 qcom_uart_wakeup_config.....	151
7.4.39 qcom_mem_alloc	151
7.4.40 qcom_mem_free.....	152
7.4.41 qcom_ap_hidden_mode_enable	152
7.4.42 qcom_ap_set_max_station_number.....	152
7.4.43 qcom_ap_set_flag	152
7.4.44 qcom_op_set_mode	153
7.4.45 qcom_op_get_mode	153
7.4.46 qcom_disconnect.....	153
7.4.47 qcom_get_disconnect_reason.....	154
7.4.48 qcom_set_ap_country	154
7.4.49 qcom_get_country_code	154
7.4.50 qcom_set_phy_mode	155
7.4.51 qcom_get_phy_mode	155
7.4.52 qcom_set_channel	155
7.4.53 qcom_get_channel	155
7.4.54 qcom_set_tx_power	155
7.4.55 qcom_get_tx_power	156
7.4.56 qcom_allow_aggr_set_tid.....	156
7.4.57 qcom_allow_aggr_get_tid.....	156
7.4.58 qcom_scan_set_mode.....	156
7.4.59 qcom_scan_get_mode	157
7.4.60 qcom_get_state	157
7.4.61 qcom_sec_set_wepkey	157
7.4.62 qcom_sec_get_wepkey	157
7.4.63 qcom_sec_set_wepkey_index.....	158
7.4.64 qcom_sec_get_wepkey_index.....	158

7.4.65 qcom_sec_set_auth_mode.....	158
7.4.66 qcom_sec_get_auth_mode	159
7.4.67 qcom_sec_set_encrypt_mode.....	159
7.4.68 qcom_sec_get_encrypt_mode.....	159
7.4.69 qcom_sec_set_passphrase	159
7.4.70 qcom_sec_get_passphrase.....	160
7.4.71 qcom_power_set_mode	160
7.4.72 qcom_power_get_mode	160
7.4.73 qcom_suspend_enable.....	160
7.4.74 qcom_suspend_start	161
7.4.75 qcom_power_set_parameters	161
7.4.76 qcom_promiscuous_enable.....	162
7.4.77 qcom_set_promiscuous_rx_cb	162
7.4.78 qcom_get_bssid	162
7.4.79 qcom_scan_params_set.....	162
7.4.80 qcom_param_set.....	163
7.4.81 qcom_mcast_filter_enable.....	164
7.4.82 qcom_set_bmiss_time.....	164
7.4.83 qcom_set_vendor_specific_ie_cmd.....	164
7.4.84 qcom_ani_enable	165
7.5 WPS APIs	165
7.5.1 WPS data structure	165
7.5.2 qcom_wps_enable.....	167
7.5.3 qcom_wps_start	167
7.5.4 qcom_wps_stop.....	167
7.5.5 qcom_wps_connect.....	168
7.5.6 qcom_wps_register_event_handler	168
7.5.7 qcom_wps_set_credentials	168
7.5.8 qcom_wps_event_handler_t.....	168
7.6 P2P APIs	169
7.6.1 qcom_p2p_func_init	169
7.6.2 qcom_p2p_func_cancel.....	169
7.6.3 qcom_p2p_func_set_pass_ssid	169
7.6.4 qcom_p2p_func_get_pass_ssid	170
7.6.5 qcom_p2p_func_auth	170
7.6.6 qcom_p2p_func_connect.....	170
7.6.7 qcom_p2p_func_start_go	171
7.6.8 qcom_p2p_func_set_config.....	171
7.6.9 qcom_p2p_func_invite.....	171
7.6.10 qcom_p2p_func_join	172
7.6.11 qcom_p2p_func_prov	172
7.6.12 qcom_p2p_func_set_opps.....	172
7.6.13 qcom_p2p_func_set_noa	173
7.6.14 qcom_p2p_func_find	173
7.6.15 qcom_p2p_func_stop_find.....	173
7.6.16 qcom_p2p_func_get_node_list.....	173
7.6.17 qcom_p2p_func_get_network_list	174
7.6.18 qcom_p2p_func_set	174
7.6.19 qcom_p2p_func_invite_auth.....	174
7.6.20 qcom_p2p_func_wps_start_no_scan	175
7.6.21 qcom_p2p_func_listen.....	175
7.6.22 qcom_p2p_get_role.....	175
7.6.23 qcom_p2p_event_handler	176
7.6.24 qcom_p2p_func_register_event_handler.....	176
7.7 SSL APIs	176
7.7.1 qcom_SSL_CTX_new	176
7.7.2 qcom_SSL_CTX_free.....	176
7.7.3 qcom_SSL_new	177
7.7.4 qcom_SSL_accept	177
7.7.5 qcom_SSL_connect	177
7.7.6 qcom_SSL_configure	177
7.7.7 qcom_SSL_context_configure.....	178

7.7.8 qcom_SSL_setCaList	178
7.7.9 qcom_SSL_addCert	178
7.7.10 qcom_SSL_storeCert	178
7.7.11 qcom_SSL_loadCert	179
7.7.12 qcom_SSL_listCert	179
7.7.13 qcom_SSL_read	179
7.7.14 qcom_SSL_set_fd	179
7.7.15 qcom_SSL_shutdown	180
7.7.16 qcom_SSL_write	180
7.7.17 qcom_SSL_set_tm	180
7.8 Hardware APIs	180
7.8.1 qcom_gpio_apply_peripheral_configuration	180
7.8.2 qcom_gpio_add_alternate_configurations	181
7.8.3 qcom_gpio_peripheral_pin_conflict_check	181
7.8.4 qcom_gpio_pin_dir	182
7.8.5 qcom_gpio_pin_set	182
7.8.6 qcom_gpio_get_interrupt_pin_num	182
7.8.7 qcom_gpio_interrupt_register	183
7.8.8 qcom_gpio_interrupt_mode	184
7.8.9 qcom_gpio_interrupt_wakeup	184
7.8.10 qcom_gpio_pin_source	185
7.8.11 qcom_i2cm_init	185
7.8.12 qcom_i2cm_read	185
7.8.13 qcom_i2cm_write	186
7.8.14 qcom_i2cm_fini	186
7.8.15 qcom_i2cs_control	186
7.8.16 qcom_i2cs_cmd	187
7.8.17 qcom_i2cs_csr_init	187
7.8.18 qcom_i2cs_reg_init	187
7.8.19 qcom_i2cs_fifo_init	187
7.8.20 qcom_pwm_control	187
7.8.21 qcom_pwm_sdm_set	188
7.8.22 qcom_pwm_port_set	188
7.8.23 qcom_spi_init	189
7.8.24 qcom_spi_fini	189
7.8.25 qcom_spi_wait_done	189
7.8.26 qcom_spi_request	190
7.8.27 qcom_spi_response	190
7.8.28 qcom_i2s_init	190
7.8.29 qcom_i2s_rcv_control	191
7.8.30 qcom_i2s_rcv_data	191
7.8.31 qcom_i2s_xmt_data	191
7.8.32 qcom_adc_init	192
7.8.33 qcom_adc_config	192
7.8.34 qcom_adc_timer_config	193
7.8.35 qcom_adc_dma_config	193
7.8.36 qcom_adc_calibration	193
7.8.37 qcom_adc_conversion	193
7.8.38 qcom_adc_rcv_data	194
7.8.39 qcom_adc_close	194
7.8.40 qcom_wkup_pin_config	194
7.8.41 qcom_uart_rx_pin_set	194
7.8.42 qcom_uart_tx_pin_set	195
7.8.43 qcom_dset_create	195
7.8.44 qcom_dset_open	195
7.8.45 qcom_dset_read	196
7.8.46 qcom_dset_write	196
7.8.47 qcom_dset_close	197
7.8.48 qcom_dset_commit	197
7.8.49 qcom_dset_delete	197
7.8.50 qcom_dset_size	198
7.8.51 qcom_dset_media	198

7.9 Unified Security APIs	198
7.9.1 Key management APIs	198
7.9.2 Cryptographic operations API	209
7.9.3 Cryptographic algorithms specification	222
7.10 Miscellaneous APIs	227
7.10.1 qcom_sys_reset	227
7.10.2 qcom_mac_get	227
7.10.3 qcom_timer_init	227
7.10.4 qcom_timer_start	228
7.10.5 qcom_timer_stop	228
7.10.6 qcom_timer_delete	228
7.10.7 qcom_timer_us_start	228
7.10.8 qcom_time_us	228
7.10.9 qcom_watchdog	229
7.10.10 qcom_watchdog_feed	229
7.10.11 qcom_suspend_restore_flag_get	229
7.10.12 qcom_enable_print	229
7.10.13 qcom_vsnprintf	229
7.10.14 qcom_console_get_free_txbuf_sz	230
7.10.15 qcom_aes_encrypt_init	230
7.10.16 qcom_aes_encrypt	230
7.10.17 qcom_aes_encrypt_deinit	230
7.10.18 qcom_aes_decrypt_init	231
7.10.19 qcom_aes_decrypt	231
7.10.20 qcom_aes_decrypt_deinit	231
7.10.21 qcom_time_us	231
7.10.22 qcom_isr_handler_install	231
7.10.23 qcom_cust_speed_ctrl	232
A P2P Linux Client Commands	233
A.1 Operation groups overview	233
A.2 wpa_cli	233
A.3 Device discovery	233
A.3.1 p2p_find	233
A.3.2 p2p_listen	234
A.3.3 p2p_stop_find	234
A.3.4 p2p_flush	234
A.4 Group formation	234
A.4.1 p2p_prov_disc	234
A.4.2 p2p_connect	234
A.4.3 p2p_group_add	235
A.4.4 p2p_reject	235
A.4.5 p2p_group_remove	236
A.4.6 p2p_cancel	236
A.4.7 p2p_remove_client	236
A.5 Invitation	236
A.5.1 p2p_invite	236
A.6 Group operations	237
A.6.1 wps_pin	237
A.6.2 wps_pbc	237
A.6.3 p2p_get_passphrase	237
A.6.4 p2p_presence_req	237
A.7 Parameters	237
A.7.1 p2p_ext_listen	237
A.7.2 p2p_set	238
A.7.3 set238	
A.8 Status	239
A.8.1 p2p_peers	239
A.8.2 p2p_peer	239
A.9 Group status	239

A.9.1 status.....	240
A.9.2 sta240	
A.9.3 all_sta.....	240
A.9.4 list_network.....	240
A.9.5 remove_network.....	240

Figures

Figure 2-1 Routing UART interfaces in hostless mode for RB01/RB02	17
Figure 2-2 Routing interfaces in JTAG mode for RB01/RB02	17
Figure 2-3 Routing USB interface in calibration mode for RB01/RB02	18
Figure 3-1 Example system configuration (RB01+RB02).....	20
Figure 3-2 HTTP GET method	50
Figure 3-3 OTA firmware upgrade process	57
Figure 3-4 GPIO data structure after configuration	72
Figure 4-1 Sample system configuration for P2P demonstration (RB01+RB02)	82
Figure 6-1 Hardware rework of RB01/RB02 for power measurement	107
Figure 6-2 Function jumper configuration of RB01 for power measurement	107
Figure 6-3 Power measurement points for RB01/RB02	108
Figure 7-1 System memory allocation	115
Figure 7-2 Address allocation for ROM and RAM	116

Tables

Table 3-1 Configuration component command usage.....	22
Table 3-2 Benchmark commands	44
Table 3-3 GPIO configurations.....	67
Table 3-4 GPIO active/inactive configuration	68
Table 4-1 P2P command options	75
Table 6-1 Power measurement environment setup.....	106
Table 7-1 Usage constants	199
Table 7-2 Handle flag constants.....	200
Table 7-3 Operation constants	200
Table 7-4 Operation states.....	200
Table 7-5 Object types and key sizes	203
Table 7-6 Supported attributes.....	205
Table 7-7 Object types and key sizes	207
Table 7-8 Supported cryptographic algorithms	209
Table 7-9 ECC cryptographic algorithms	209
Table 7-10 Possible QCOM_CRYPTOMODE values	210
Table 7-11 qcom_crypto_op_alloc allowed modes	211
Table 7-12 Public key allowed modes	213
Table 7-13 Key-Pair parts for operation modes.....	213
Table 7-14 Asymmetric derivation operation parameters.....	221
Table 7-15 List of algorithm identifiers	222
Table 7-16 Structure of algorithm identifier	224
Table 7-17 List of object types.....	224
Table 7-18 List of supported ECC curves	225
Table 7-19 Object or operation attributes	225
Table 7-20 Attribute format definitions	227

1 Introduction

This document provides instructions for setting up the Qualcomm Technologies IoE development kit (hereafter referred to as *development kit*), as well as for building and using the hostless SDK and the P2P demo application. The hostless SDK supports the RB01/RB02 reference design (based on the Qualcomm Technologies, Inc. QCA4010/QCA4012 WLAN SoC). The following applications are provided with the software release:

- SDK shell
 - Wi-Fi
 - HTTP server/client
 - DNS server/client
 - SNTP
 - SSL
 - WPS
 - OTA firmware upgrade
- P2P demo applications



Figure 1-1 Front view of RB01/RB02

Acronyms and abbreviations

Acronym or abbreviation	Definition
AP	Access point
DIP	Dual in-line package
DNS	Domain name system
OTA	Over-the-air
OTP	One-time programmable
SDK	Software development kit
SNTP	Simple network time protocol
SoC	System on a chip
SSL	Secure socket layer
WPS	Wi-Fi protected setup

2 Board Setup

This chapter describes the interface routings based on the DIP switch and jumper settings. Many other interface combinations are user-defined and are based on the schematics and chip specifications. Due to multiplexing of GPIO pins, some interfaces or peripherals cannot be enabled with others. Refer to *QCA4010 Device Specification* and *QCA4012 Device Specification* for details.

2.1 RB01/RB02 board setup

2.1.1 Access UART interface in hostless mode

The RB02 module has two UART interfaces: Debug UART and HSAURT. For debug UART the header is connected through a USB2RS232 adapter cable to the serial port on a PC. For HSUART the header is connected through fly wire. Refer to *RB01 Development Platform Hardware User Guide* for detailed interface assignments and definitions.

Pull on the jumper connecting JP3.1&2 (for RB02), JP5.2&3(IOT mode), JP11.2&1(Test mode), JP10.2&3(HOST0), and JP9.2&1(HOST1) to place the RB02 board in hostless mode. Refer Figure 2-1.

Interface	Signal	Header pins
UART1	QCA4010/QCA4012 RXD	J1.6
	QCA4010/QCA4012 GND	J1.7
	QCA4010/QCA4012 TXD	J1.8
HSUART	QCA4010/QCA4012 TXD	JP6.4
	QCA4010/QCA4012 GND	JP6.5
	QCA4010/QCA4012 RXD	JP6.6
	QCA4010/QCA4012 CTS	JP6.8
	QCA4010/QCA4012 RTS	JP6.10

NOTE: The labels for TXD (J1.6) and RXD (J1.8) on the board are opposite to the actual settings.

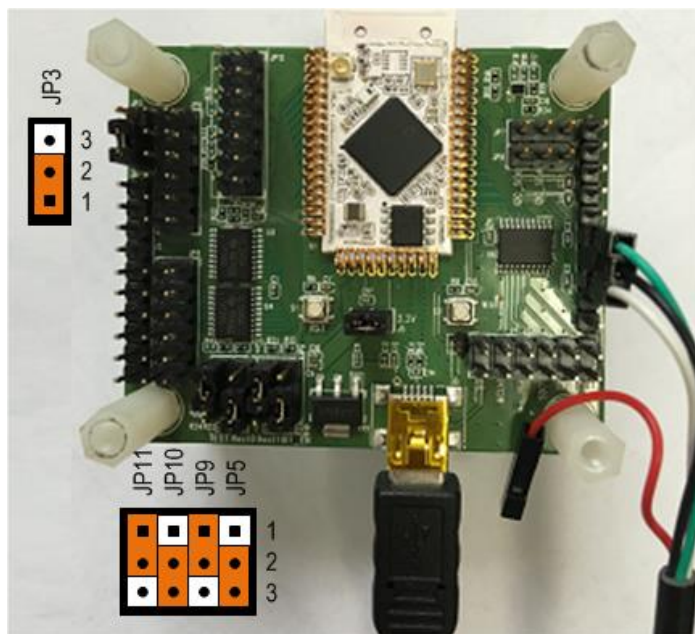


Figure 2-1 Routing UART interfaces in hostless mode for RB01/RB02

2.1.2 Access interfaces in JTAG mode

To enable JTAG mode on the RB01/RB02 platform, pull on the jumper connecting JP3.1&2 (for RB02), JP5.2&3(IOT mode), and JP11.2&3(Test mode) on RB01.

The EJTAG connector is JP12.

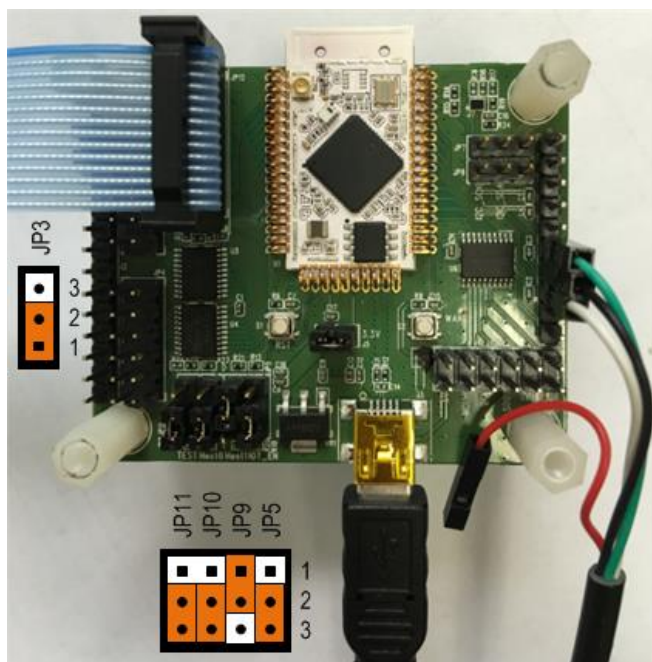


Figure 2-2 Routing interfaces in JTAG mode for RB01/RB02

2.1.3 Access USB interface in calibration mode

Pull on the jumper connecting JP3.1&2 (for RB02), JP5.2&3(IOT mode), JP11.2&1(Test mode), JP10.2&1(HOST0), and JP9.2&1(HOST1) to place the RB02 board in USB mode. The USB connector is J3.

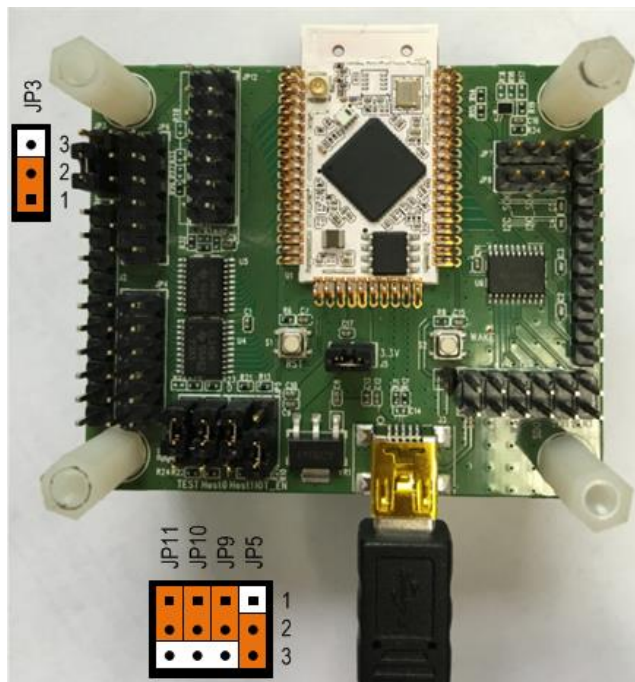


Figure 2-3 Routing USB interface in calibration mode for RB01/RB02

3 Hostless SDK Shell

The QCA4010/QCA4012 SDK contains a shell and a PC application, `ath_console.exe`, which enable software developers to configure and evaluate the performance and power consumption of the QCA4010/QCA4012 device. The SDK shell is equipped to perform basic system-level throughput testing only and is not a full-featured performance tool. The SDK shell also includes mechanisms to set QCA4010/QCA4012 to various power states for evaluating power consumption. The required equipment includes:

- Development kit
- PC with Microsoft Windows operating system
- Throughput demo endpoint application (**`ath_console.exe`**)
- Wi-Fi Access Point

The throughput demo application includes two components that can be accessed from the serial port console:

- Configuration component: Provides an interface to configure and control the behavior of the QCA4010/QCA4012 Wi-Fi SoC. This component exposes a set of configuration commands that can be used to configure and connect to an existing network.
- Benchmark component: Provides a set of commands to run bidirectional throughput streams to analyze network throughput performance. TCP traffic and UDP traffic are supported. IPv4 and IPv6 modes are supported.

The **`ath_console.exe`** application (directory: `target\tool\windows\ath_console.exe`), which can be run on a PC, also supports the benchmark component (command set) on its console.

3.1 Hostless system setup

The setup includes a development kit connected to a PC through the usbWiggler ONCE (JP12), power port (J3), and a console port (J1). A console application running on the Wi-Fi module is used to input commands for configuration and benchmarking. TCP/UDP streams can be set up between a client application (**`ath_console.exe`**) running on a PC and the hostless API running on a Wi-Fi module over a Wi-Fi link.

[Figure 3-1](#) shows a configuration example where the AP subnet is 192.168.0.x. In a real test environment where a different subnet might be used, set the correct IP addresses for the endpoints.

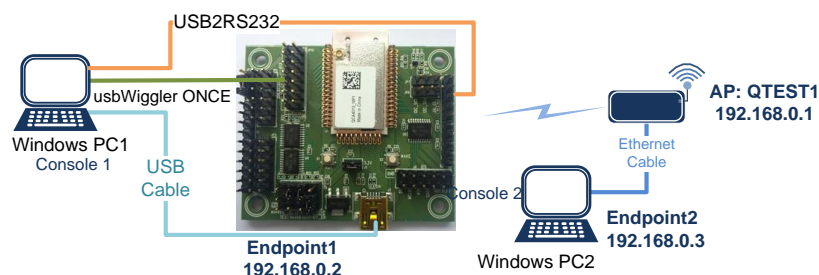


Figure 3-1 Example system configuration (RB01+RB02)

Windows PC1 setup

1. Plug the usbWiggler ONCE adapter into PC1.
2. Connect the usbWiggler ONCE adapter cable to JP12 on the RB01/RB02 development platform.
3. Use a USB2RS232 adapter cable to connect PC1 and the RB01/RB02 through J1.
4. Use a USB cable to connect PC1 and the RB01/RB02 through J3.

Wait several seconds for the RB01/RB02 to start up.

5. Start a serial terminal application from PC1 and select the lower COM port to connect using the port setting: 115200, 8, n, 1, no flow control.

Windows PC2 setup

6. Use the Ethernet cable to connect to the AP and configure the IP address to 192.168.0.3.
7. Run `ath_console.exe` as Console2 at the windows DOS prompt.

3.2 Hostless SDK shell for throughput and power measurement

3.2.1 Measuring throughput

Measure throughput with the TCP uplink and downlink tests by typing the commands, in the order shown (for example, on Console 2 then Console 1 for the TCP uplink test).

TCP uplink test	
Console2	<code>ath_console.exe rx 192.168.0.3 2390 tcp v4</code>
Console1	<code>wmiconfig --connect Qtest-1</code> <code>wmiconfig --ipstatic 192.168.0.2 255.255.255.0 192.168.0.1</code> <code>benchtx 192.168.0.3 2390 tcp 1400 0 20 0</code>
TCP downlink test	
Console1	<code>wmiconfig --connect Qtest-1</code> <code>wmiconfig --ipstatic 192.168.0.2 255.255.255.0 192.168.0.1</code> <code>benchrx tcp 2390</code>

Console2	ath_console.exe tx 192.168.0.2 2390 tcp 1400 0 20 0 v4
----------	---

3.2.2 Measuring power

The Wi-Fi module provides multiple power supply configuration options. Power measurement procedures are described in section 6.

3.2.2.1 Operating modes and data capture

Power consumption of the QCA4010/QCA4012 SoC depends on the duty cycle between various operating states of the WLAN subsystem. The QCA4010/QCA4012 firmware provides an API to set the WLAN subsystem to various operating modes. This section describes the operating modes for power profiling purposes and the associated API calls required to enable the modes.

Operating mode	Suspend QCA4010/QCA4012 is powered down, the chip shuts down all circuits except a few critical blocks needed to resume operation after being suspended.	
Description	Suspend mode turns off almost all circuitries in QCA4010/QCA4012 except the Wakeup Manager and PMU circuits. To use the Suspend feature correctly, observe the following rules: <ul style="list-style-type: none"> ▪ The suspend feature must be enabled before the QCA4010/QCA4012 associates with an AP using the command: wmiconfig --suspend ▪ The suspend operation can be started any time after the suspend feature is enabled using the command: wmiconfig --suspendstart <time in ms> 	
Procedure	<ol style="list-style-type: none"> 1. Remove the jumpers and connect them to the current measurement instrument. 2. Issue the WMI commands in this table to enter power down mode. 	
WMI commands	Power down mode can be invoked by issuing these WMI commands: wmiconfig --pwrmode 0 wmiconfig --suspend wmiconfig --connect <ssid> wmiconfig --suspendstart <time in ms>	

Operating mode	Sleep QCA4010/QCA4012 WLAN blocks are powered down and the clocks are turned off.	
Setup	<ol style="list-style-type: none"> 1. Remove the jumpers and connect them to current measurement instrument. 2. Issue the WMI commands from this table to enable Sleep mode. 	
WMI commands	Sleep mode can be invoked by issuing these WMI commands:	
	wmiconfig --disc	Disconnect from the AP
	wmiconfig --pwrmode 1	Enable low power mode

Operating mode	Associated QCA4010/QCA4012 is duty cycling between the sleep state and active WLAN Rx and WLAN Tx.	
Description	The associated mode is enabled by allowing QCA4010/QCA4012 to connect to an AP with periodic wakeup and listening for beacons.	
Setup	<ol style="list-style-type: none"> 1. Remove the jumpers and connect them to the current measurement instrument. 2. Issue the WMI commands from this table to connect to the AP and enable sleep mode. 	

WMI commands	This sequence of WMI commands is an example of how to enable this power mode. The command <code>wmiconfig --connect <ssid></code> connects to the open mode AP.	
	<code>wmiconfig --connect <ssid></code>	Set up the SSID of the AP
	<code>wmiconfig --pwrmode 1</code>	Enable low power mode

3.3 Console command reference

This section describes the configuration commands the throughput tool supports.

Table 3-1 Configuration component command usage

Command	Description
help	Display the help menu and syntax of available commands.
version	Display firmware and driver version strings.
connect	Establish a connection to an AP, specified by its SSID, in open mode.
wepkey	Connect to an AP in WEP mode.
passphrase	Connect to an AP in WPA/WPA2 mode.
wps	Setup and start a WPS operation.
disc	Disconnect from the current AP (or a peer in case of ad hoc).
settxpower	Set the transmit power level.
pwrmode	Set the device power mode.
channel	Set the channel hint to have the device try to connect on the specified channel first.
wmode	Set Wi-Fi mode.
listen	Set the listen interval.
rssi	Display the link quality of the downlink between the AP and the STA.
pmparams	Set the power management parameters.
scanctrl	Enable/disable foreground/background scanning.
setscanpara	Set the scan parameters.
setscan	Set parameters before starting a long or short channel scan.
driver	Unload/reload the Wi-Fi driver.
allow_aggr	Control aggregation behavior.
iwconfig scan	Scan for APs.
reset	Reset the board.
ipconfig	Display the target stack IP parameters.
ipstatic	Assign static IP information to the IP stack.
ipdhcp	Start the IPv4 DHCP client in the IP stack.
mode	Configure the device to the SoftAP.
ap bconint	Set the SoftAP beacon interval.
ap setmaxstanum	Set the maximum number of stations that the AP supports.
ipdhcppool	Set the address pool for the DHCP server.
ipv6rtprfx	Set the IPv6 router prefix.
ip_dns_client	Start/Stop the DNS client at run time.
ip_dns_server	Start/Stop the DNS server.
ip_dns_server_addr	Set the DNS server address.

Command	Description
ip_dns_local_domain	Set the local domain name for the DNS Server.
ip_dns	Create a DNS entry for the DNS server.
ip_dns	Delete the DNS entry for the DNS server.
ip_dns_delete_server_addr	Delete the DNS Server for the DNS Client.
ip_resolve_hostname	Resolve the host name by domain type.
ip_gETHOSTbyname	Resolve the host name (for IPv4 only).
ip_gETHOSTbyname2	Resolve the hostname by domain type.
ipbridgemode	Enable bridging mode in the firmware.
tcp_backoff_retry	Set the TCP backoff retry limit.
settcptimeout	Set the TCP connection timeout value in seconds.
setmainttimeout	Set the maintenance timer timeout.
ip_sntp_client	Start/Stop the SNTP client.
ip_sntp_srvr	Start/Stop the SNTP server.
ip_sntp_get_time	Get the current timestamp from the SNTP server.
ip_sntp_get_time_of_day	Display the time in seconds.
ip_sntp_zone	Modify the client zone and enable/disable daylight saving.
ip_show_sntpconfig	Display the SNTP server address.
ip_http_server	Start/stop the HTTP Server at run time.
ip_http_set_custom_uri	Set the customer URI received for the HTTP server.
ip_http_redirect_unknown_url	Enable/disable unknown URL redirection function.
ip_http_server_add_redirected_html	Add the server-side customer-defined redirected HTML.
ip_http_set_redirected_url	Set the customer defined URL for redirecting HTML.
ip_http_restrict_http_request	Enable/disable the restriction function of the HTTP client request.
ip_http_post	Post or update the object to the HTML page.
ip_http_get	Get the updated database object of a given HTML page.
ip_http_client	Configure the HTTP client at run time.
ssl_start	Start the SSL as either a server or a client.
ssl_stop	Stop the SSL as either a server or a client.
ssl_config	Configure the SSL server or client.
ssl_add_cert	Add a certificate or a CA list to the SSL server or client.
ssl_store_cert	Store a certificate or a CA list to the flash.
ssl_delete_cert	Delete a certificate or CA list by name from flash.
ssl_list_cert	List the names of the certificates and CA lists stored in the flash.
wdt	Enable/Disable the watch dog timer.
device	Select the device ID to which the subsequent wmicconfig commands take effect on.
ota_upgrade	Initiate an OTA upgrade service.
ota_read	Read OTA image bytes from the OTA area.
ota_done	Terminate and close the OTA service.
pwm	PWM test with 8 channels.
i2c	I ² C master/slave mode read/write test.

Command	Description
i2s	I ² S playback start/stop or I ² S recorder start/stop.
pingld	Set the Ping ID of the Ping packet.
ip_dns_timeout	Set the DNS timeout value.
mcastfilter	Enable/disable the multicast filter.
print	Enable/disable the debug information inside firmware.
ota_ftp	OTA FTP upgrade.
ota_http	OTA HTTP upgrade.
ota_https	OTA HTTPS upgrade.
bmiss	Configure the threshold for WLAN STA to report beacon miss.
appleie	Enable/disable a sample Apple information element of management frames.
ani	Enable/Disable ANI.
partition_index	Get the firmware partition index from which system booted.
timer_socket	Start/stop the 15.625 ms timer to send or echo UDP data.

3.3.1 help

Syntax	<i>wmiconfig --help</i>
Description	Display the help menu and syntax of available commands
Parameters	None

3.3.2 version

Syntax	<i>wmiconfig --version</i>
Description	Display the firmware and driver versions
Parameters	None

3.3.3 connect

Syntax	<i>wmiconfig --connect <ssid></i>		
Description	Establish a connection to an AP, specified by its SSID, in open mode. In case of successful connection, the connected message is printed on the console. In case of failure, the message not connected is printed.		
Parameters	<table border="1"> <tr> <td><ssid></td><td>Network SSID</td></tr> </table>	<ssid>	Network SSID
<ssid>	Network SSID		

3.3.4 wepkey

Syntax	<i>wmiconfig --wepkey <key_index> <key></i> <i>wmiconfig --wep <def_keyix> [mode]</i> <i>wmiconfig --connect <ssid></i>
---------------	---

Description	Connect to an AP in WEP mode. The WEP connection is done in two steps: first setting the WEP key at the specified key index, then issuing the WEP configuration command with WEP parameters to trigger a connection. All parameters are mandatory. The WEP key is either 10 or 26 hexadecimal characters.	
Parameters	<key_index>	WEP key index [1-4]
	<key>	WEP key
	<ssid>	Network SSID
	[mode]	Set to open or shared or auto
	<def_keyix>	Default WEP key index [1-4]
Example	<pre> wmiconfig --wepkey 1 001122aabb wmiconfig --wep 1 open wmiconfig --connect iot </pre>	

3.3.5 passphrase

Syntax	<pre> wmiconfig --p <passphrase> wmiconfig --wpa <ver> <ucipher> <mcipher> wmiconfig --connect <ssid> </pre>	
Description	Connect to an AP in WPA/WPA2 mode. The WPA/WPA2 connection is done in two steps: first setting the passphrase, then using the WPA configuration command to configure the WPA type and ciphers. The last command triggers a connection with WPA parameters. All parameters are mandatory.	
Parameters	<passphrase>	Passphrase
	<ssid>	Network SSID
	<ver>	1 WPA
		2 WPA2
	<uciper>	TKIP or CCMP
	<mciper>	TKIP or CCMP
Example	<pre> wmiconfig --p passphrase123 wmiconfig --wpa 1 TKIP TKIP wmiconfig --connect iot </pre>	

3.3.6 wps

Syntax	<i>wmiconfig --wps <connect> <mode> [pin] [ssid] [mac] [channel]</i>	
Description	Set up and start a WPS operation. Both PUSH and PIN modes are supported. The last three parameters are optional; but if one is specified, the other two must be specified as well. All parameters obtained via WPS are printed on the screen.	
Parameters	<connect>	0 Does not attempt to connect after WPS
		1 Attempts to connect after WPS
	<mode>	Set to push or pin
	[pin]	(Optional) Pin code
	[ssid]	(Optional) Specific SSID for WPS handshake
	[mac]	(Optional) Specific MAC address for WPS handshake
	[channel]	(Optional) Specific channel ID

Example	<code>wmiconfig --wps 1 pin 2235341</code>
----------------	--

3.3.7 disc

Syntax	<code>wmiconfig --disc</code>
Description	Disconnect from the current AP (or peer in case of ad hoc)
Parameters	None

3.3.8 settxpower

Syntax	<code>wmiconfig --settxpower <val></code>		
Description	Set the transmit power level. Transmit power settings must not exceed the limit by regulatory domain for the country that has been selected. Intended for test and evaluation. Allowing the device to determine the optimal Tx output power, based on rate, is recommended.		
Parameters	<code><val></code>	0	Power value from 0 to 63. Default is 63. Refer to the related Device Specifications for the maximum Tx Power in dBm.

3.3.9 pwrmode

Syntax	<code>wmiconfig --pwrmode <val></code>		
Description	Set the device power mode. Default mode is 1.		
Parameters	<code><val></code>	0	High performance
		1	Power save

3.3.10 channel

Syntax	<code>wmiconfig --channel <channel></code>		
Description	Set the channel hint to have the device try to connect on the specified channel first. This command must be issued before the connect command. If WPS is enabled, the channel detected during the exchange is used. If AP is not found on the specified channel, the device tries connecting on other channels.		
Parameters	<code><channel></code>	Specified channel	

3.3.11 wmode

Syntax	<code>wmiconfig --wmode <b g n ht40 [<above below>]>a [<ht20 ht40>]></code>		
Description	Set the Wi-Fi mode.		
Parameters	a	802.11a Works for the 5G band. NOTE: Parameter ht20 or ht40 is valid only when mode = a. If the parameter ht20 is set, the DUT runs in the HT20 mode. If the parameter ht40 is set, the DUT runs in the HT40 mode. If no parameter is set, the DUT works in the pure 11a mode.	

	b	802.11b
	g	802.11g
	n	802.11n Works for the 2.4G band in the HT20 mode.
	ht40	Works for the 2.4G band in the 11n HT40 mode. NOTE: Parameter above or below is valid for ht40 only. If the parameter above is set, the second channel is home channel + 4. If the parameter below is set, the second channel is home channel - 4. If no parameter is set, it defaults to above.
Example	wmiconfig --wmode ht40 above	

3.3.12 listen

Syntax	<i>wmiconfig --listen <interval></i>	
Description	Set the listen interval (15 to 5000 TUs). 1 TU is approximately 1 ms. Default is 100 TU (ms). This command must be issued before the connect command.	
Parameters	<interval>	Listen Interval in TUs [15-5000]: default 100 TUs

3.3.13 rssi

Syntax	<i>wmiconfig --rssi</i>	
Description	Display the link quality of the downlink between AP and STA. Print the link quality (SNR) in dB. This command is used only when the STA is associated to an AP and downlink data is received. To obtain Beacon RSSI, the scan command can be used (iwconfig scan).	
Parameters	None	

3.3.14 pmparms

Syntax	<i>wmiconfig --pmparms --idle <msec> --np <n> --dp <1-4> --txwp <1-2> --ntxw <> --psfp <1-2></i>	
Description	Set the power management parameters.	
Parameters	--idle < >	Idle period in ms (default is 200)

	--np < >		The number of power save poll messages sent from the device before notifying the AP that it is awake. Default is 10.	
	--dp < >		DTIM Policy (default is Stick)	
			1	Ignore (device does not listen to any traffic after beacon)
			2	Normal (wakeup only at TIM interval)
			3	Stick (wakeup only at DTIM interval)
			4	Auto (wakeup at both TIM and DTIM intervals)
	--txwp < >		Tx wakeup policy. Policy to determine whether Tx can wake up WLAN. (Default is Do not wake)	
			1	Wake upon sleep (indicate awake to AP upon uplink traffic).
			2	Do not wake (never transit to awake by uplink traffic).
	--ntxw < >		Number of uplink frames in a beacon interval to transit to awake (Default = 1).	
	--psfp < >		Power save failure event policy. Policy to determine if power save failure event is sent to host during scanning. (Default = Ignore event)	
			1	Send fail event.
			2	Ignore event.
Example		<i>wmiconfig --pmpparams --idle 100 --np 1 --dp 3 --txwp 1 --ntxw 1 --psfp 1</i>		

3.3.15 scanctrl

Syntax	<i>wmiconfig --scanctrl <foreground></i>		
Description	Enable/disable foreground/background scanning.		
Parameters	<i><foreground></i>	0	Disable foreground scan.
		1	Enable foreground scan.

3.3.16 setscanpara

Syntax	<i>wmiconfig -- setscanpara <max_act_ch_dwell_time_ms> <pas_act_chan_dwell_time_ms> <fg_start_period> <fg_end_period> <bg_period> <short_scan_ratio> <scan_ctrl_flags> <min_active_chan_dwell_time_ms> <max_act_scan_per_ssid> <max_dfs_ch_act_time_in_ms></i>		
Description	Set scan parameters		
Parameters	<i>max_act_ch_dwell_time_ms</i>	Set max dwell time in ms (0=reset value)	
	<i>pas_act_chan_dwell_time_ms</i>	Passive channel dwell time in ms (0=reset value)	
		Enable background scan.	

	<i>fg_start_period</i>	fg scan start period in second (0=reset value, 65535=disable)
	<i>fg_end_period</i>	fg scan end period in second (0=reset value)
	<i>bg_period</i>	bg scan period in second (0=disable, 65535=disable)
	<i>short_scan_ratio</i>	Short scan ratio (default=3)
	<i>scan_ctrl_flags</i>	Scan control flags (default=47)
	<i>min_active_chan_dwell_time_ms</i>	Min active chan dwell time in ms
	<i>max_act_scan_per_ssid</i>	Max scan per SSID
	<i>max_dfs_ch_act_time_in_ms</i>	Max time dfs chan active in ms

3.3.17 setscan

Syntax	<i>wmiconfig --setscan <forceFgScan> <homeDwellTimeInMs> <forceScanIntervallInMs> <scanType> <numChannels> [<channel> <channel>... up to numChannels]</i>		
Description	Set parameters before starting a long or short channel scan.		
Parameters	<i>forceFgScan</i>	0	Disable foreground scan.
		1	Enable foreground scan.
	<i>homeDwellTimeInMs</i>	Home chan dwell time in ms (0=default)	
	<i>forceScanIntervallInMs</i>	Force scan interval in ms	
	<i>scanType</i>	Scan type as long or short scan	
	<i>numChannels</i>	numchannels to scan (0=no channels provided)	

3.3.18 driver

Syntax	<i>wmiconfig --driver <val></i>		
Description	Unload/reload the Wi-Fi driver.		
Parameters	<val>	down	Unload the driver.
		up	Reload the driver.

3.3.19 allow_aggr

Syntax	<i>wmiconfig --allow_aggr <tx_tid_mask> <rx_tid_mask></i>		
Description	Control aggregation behavior. Enable aggregation based on the provided bit mask where each bit represents a TID. Valid TID range: 0x0 to 0x7.		
Parameters	<tx_tid_mask>	Transmission TID mask (0x0 – 0xFF)	
	<rx_tid_mask>	Reception TID mask (0x0 – 0xFF)	

3.3.20 iwconfig scan

Syntax	<i>iwconfig scan [ssid]</i>		
Description	Scan for APs. If an SSID is specified, scan only for the specified SSID.		
Parameters	[ssid]	Optional parameter that forces a scan on a given SSID	

3.3.21 reset

Syntax	<i>wmiconfig --reset</i>
Description	Reset the board.
Parameters	None

3.3.22 ipconfig

Syntax	<i>wmiconfig --ipconfig</i>
Description	Display IP parameters of target stack. This command is only relevant with Offloaded IP Stack. IPv4 parameters are displayed.
Parameters	None

3.3.23 ipstatic

Syntax	<i>wmiconfig --ipstatic <IP Address> <Subnet Mask> <Default Gateway></i>	
Description	Assign static IP information to the IP stack. This command is only relevant with Offloaded IP Stack. Only IPv4 parameters can be assigned.	
Parameters	<i>IP Address</i>	IPv4 address for the interface using dot notation
	<i>Subnet Mask</i>	IPv4 Subnet mask
	<i>Default Gateway</i>	IPv4 address of Default Gateway
Example	<i>wmiconfig --ipstatic 192.168.1.90 255.255.255.0 192.168.1.1</i>	

3.3.24 ipdhcp

Syntax	<i>wmiconfig --ipdhcp</i>
Description	Start IPv4 DHCP client in IP stack. User can use the ipconfig command to obtain IP information.
Parameters	None

3.3.25 mode

Syntax	wmiconfig --mode <ap [<hidden wps>]> station>		
Description	Configure the device to SoftAP. This command only changes the mode to AP. The <i>connect</i> command is required to start SoftAP. If beacon interval and DTIM interval are different from the default values, they must set before the <i>connect</i> command. The DHCP server is enabled once the SoftAP is started.		
Parameters	ap		Set to AP mode.
		<hidden>	(Optional) Enable hidden mode in SoftAP.
		<wps>	(Optional) Enable WPS registrar mode in SoftAP.
	station	Set to Station mode.	
Example	wmiconfig --mode ap wps wmiconfig --mode ap hidden wmiconfig --mode station		

3.3.26 ap bconint

Syntax	<i>wmiconfig --ap bconint <intvl></i>	
Description	Set SoftAP beacon interval. Default is 100 ms. This command must be issued before the <i>connect</i> command.	
Parameters	<i><intvl></i>	Interval in ms

3.3.27 ap setmaxstanum

Syntax	<i>wmiconfig --ap setmaxstanum <num></i>	
Description	Set the maximum station numbers AP supports.	
Parameters	<i><num></i>	0-4

3.3.28 ipdhcppool

Syntax	<i>wmiconfig --ipdhcppool <start ipaddr> <End ipaddr> <Lease time></i>	
Description	Set the address pool for DHCP server. This command must be issued before the <i>connect</i> command.	
Parameters	<i><start ipaddr></i>	Start IP address of the DHCP address pool
	<i><End ipaddr></i>	End IP address of the DHCP address pool
	<i><Lease time></i>	DHCP lease time in seconds

3.3.29 ipv6rtpfx

Syntax	<i>wmiconfig --ipv6rtpfx <prefix> <prefixlen> <prefix_lifetime> <valid_lifetime></i>	
Description	Set IPv6 router prefix.	
Parameters	<i><prefix></i>	Set the IPv6 prefix.
	<i><prefixlen></i>	Set the IPv6 prefix length.
	<i><prefix_lifetime></i>	Set the IPv6 prefix lifetime.
	<i><valid_lifetime></i>	Set the IPv6 valid lifetime.

3.3.30 ip_dns_client

Syntax	<i>wmiconfig --ip_dns_client <start/stop></i>	
Description	Start/Stop the DNS client at run time.	
Parameters	<i><start></i>	Enable the DNS Client.
	<i><stop></i>	Disable the DNS Client.

3.3.31 ip_dns_server

Syntax	<i>wmiconfig --ip_dns_server <start/stop></i>	
Description	Start/stop DNS server.	
Parameters	<i><start></i>	Enable the DNS Server.

	<code><stop></code>	Disable the DNS Server.
--	---------------------------	-------------------------

3.3.32 ip_dns_server_addr

Syntax	<code>wmiconfig --ip_dns_server_addr <ipaddr></code>	
Description	Set the DNS server address for DNS clients.	
Parameters	<code><ipaddr></code>	IP address of the DNS server

3.3.33 ip_dns_local_domain

Syntax	<code>wmiconfig --ip_dns_local_domain <domain name></code>	
Description	Set the local domain name for the DNS Server.	
Parameters	<code><domain name></code>	Domain Name of the DNS server

3.3.34 ip_dns add

Syntax	<code>wmiconfig --ip_dns add <hostname> <ipaddr></code>	
Description	Add a DNS entry for the DNS server.	
Parameters	<code><hostname></code>	Host name of the DNS server
	<code><ipaddr></code>	IP address of the DNS server

3.3.35 ip_dns delete

Syntax	<code>wmiconfig --ip_dns delete <hostname> <ipaddr></code>	
Description	Delete the DNS entry for the DNS server	
Parameters	<code><hostname></code>	Host name of the DNS server
	<code><ipaddr></code>	IP address of the DNS server

3.3.36 ip_dns_delete_server_addr

Syntax	<code>wmiconfig --ip_dns_delete_server_addr <ipaddr></code>	
Description	Delete the DNS Server for the DNS Client.	
Parameters	<code><ipaddr></code>	IP address of the DNS Server

3.3.37 ip_resolve_hostname

Syntax	<code>wmiconfig --ip_resolve_hostname <host name> <domain type></code>	
Description	Resolves the host name by domain type.	
Parameters	<code><host name></code>	Host name
	<code><domain type></code>	Domain type 2 = IPv4 3 = IPv6

3.3.38 ip_gETHOSTbyname

Syntax	<i>wmiconfig --ip_gETHOSTbyname <host name></i>	
Description	Resolve the host name (for IPv4 only).	
Parameters	<i><host name></i>	Host name to resolve

3.3.39 ip_gETHOSTbyname2

Syntax	<i>wmiconfig --ip_gETHOSTbyname2 <host name> <domain type></i>	
Description	Resolve the hostname by the domain type.	
Parameters	<i><host name></i>	Host name
	<i><domain type></i>	Domain type 2 = IPv4 3 = IPv6

3.3.40 ipbridgemode

Syntax	<i>wmiconfig --ipbridgemode</i>	
Description	Enable the bridging mode in the firmware.	
Parameters	None	

3.3.41 tcp_backoff_retry

Syntax	<i>wmiconfig --tcp_backoff_retry <num></i>	
Description	Set the TCP backoff retry limit.	
Parameters	<i><num></i>	4-12

3.3.42 settcptimeout

Syntax	<i>wmiconfig --settcptimeout <time></i>	
Description	Set the TCP connection timeout value in seconds.	
Parameters	Time in seconds	

3.3.43 setmainttimeout

Syntax	<i>wmiconfig --setmainttimeout <time></i>	
Description	Set the maintenance timer timeout.	
Parameters	Time in milliseconds	

3.3.44 ip_snmp_client

Syntax	<i>wmiconfig --ip_snmp_client <start stop></i>	
---------------	--	--

Description	Start/stop the SNTP client.	
Parameters	<start>	Enables the SNTP Client
	<stop>	Disables the SNTP Client

3.3.45 ip_sntp_srvr

Syntax	<i>wmiconfig --ip_sntp_srvr <start stop></i>	
Description	Start/stop SNTP server.	
Parameters	<start>	Enables the SNTP Server
	<stop>	Disables the SNTP Server

3.3.46 ip_sntp_get_time

Syntax	<i>wmiconfig --ip_sntp_get_time</i>	
Description	Get current timestamp from SNTP server.	
Parameters	None	

3.3.47 ip_sntp_get_time_of_day

Syntax	<i>wmiconfig --ip_sntp_get_time_of_day</i>	
Description	Display time in seconds.	
Parameters	None	

3.3.48 ip_sntp_zone

Syntax	<i>wmiconfig --ip_sntp_zone <UTC+ -min:hr> dse <enable disable></i>	
Description	Modify client zone and enable/disable daylight saving.	
Parameters	<UTC+ -min:hr>	UTC string (example UTC+05:30)
	<enable disable>	enable: Enable daylight saving. disable: Disable daylight saving.

3.3.49 ip_show_sntpconfig

Syntax	<i>wmiconfig --ip_show_sntpconfig</i>	
Description	Display SNTP server address.	
Parameters	None	

3.3.50 ip_http_server

Syntax	<i>wmiconfig --ip_http_server <start stop></i>	
Description	Start/stop the HTTP Server at run time.	
Parameters	<start>	Enable the HTTP Server.

	<stop>	Disable the HTTP Server.
--	--------	--------------------------

3.3.51 ip_http_set_custom_uri

Syntax	<i>wmiconfig --ip_http_set_custom_uri</i>	
Description	Set the customer URI received for HTTP server	
Parameters	None	

3.3.52 ip_http_redirect_unknown_url

Syntax	<i>wmiconfig --ip_http_redirect_unknown_url <enable/disable></i>	
Description	Enable/disable unknown URL redirection function	
Parameters	<enable>	Enable redirect unknown URL function
	<disable>	Disable redirect unknown URL function

3.3.53 ip_http_server_add_redirected_html

Syntax	<i>wmiconfig --ip_http_server_add_redirected_html <html></i>	
Description	Add the server-side customer-defined redirected html	
Parameters	<html>	Newly added HTML

3.3.54 ip_http_set_redirected_url

Syntax	<i>wmiconfig --ip_http_set_redirected_url <url></i>	
Description	Set the customer-defined URL for redirecting HTML	
Parameters	<url>	Newly defined URL

3.3.55 ip_http_restrict_http_request

Syntax	<i>wmiconfig --ip_http_restrict_http_request <enable/disable></i>	
Description	Enable/disable restriction function of HTTP client request	
Parameters	<enable>	Enable restriction function in local LAN for HTTP client request
	<disable>	Disable restriction function in local LAN for HTTP client request

3.3.56 ip_http_post

Syntax	<i>wmiconfig --ip_http_post <page name> <obj_name> <obj_type> <obj_len> <obj_value></i>	
Description	Post or update the object to HTML page. This updated database is reflected on the browser for the subsequent GET requests.	
Parameters	<page name>	The name of the Page for which the data is to be updated.
	<obj_name>	Object name to update
	<Obj_type>	1= Bool; 2 = Integer; 3 = String

	<Obj_len>	Length of the object
	<Obj_value>	Object value

3.3.57 ip_http_get

Syntax	<i>wmiconfig --ip_http_get <page name> <obj_name> <obj_type></i>	
Description	Get the updated database object of a given HTML page.	
Parameters	<page name>	The name of the page from which the data is to be fetched.
	<obj_name>	Object name to get.

3.3.58 ip_http_client

Syntax	<i>wmiconfig --ip_http_client <connect get query post disc header clearheader body put patch> <data1> <data2> <ssl index></i>	
Description	Configure the HTTP client at run time.	
Parameters	<i>connect</i>	Connect to the HTTP server. Return error if any. <data1> is domain name/IP address of the HTTP server. <data2> is port number (optional). Default port is 80. <ssl index> SSL index of SSL context if connecting to HTTPS server.
	<i>get</i>	Request a page from the server. <data1> is the page name to request (mandatory).
	<i>query</i>	Update the content of page to be posted to server. Configure the (name, value) pair before POST/GET/PUT/PATCH. <data1> is the name. <data2> is the value.
	<i>post</i>	Sent a POST request to the HTTP server. <data1> is the page name to request (mandatory).
	<i>disc</i>	Disconnect the HTTP server and close the HTTP client session.
	<i>header</i>	Add customer HTTP client header buffer
	<i>clearheader</i>	Clear customer HTTP client header buffer
	<i>body</i>	Set customer HTTP client body buffer
	<i>put</i>	Send a PUT request to the HTTP server. <data1> is the page name to request (mandatory).
	<i>patch</i>	Send a PATCH request to the HTTP server. <data1> is the page name to request (mandatory).

3.3.59 ssl_start

Syntax	<i>wmiconfig --ssl_start <server client></i>	
Description	Start SSL as either server or client. Return SSL context index.	
Parameters	<server client>	Start SSL as server or client.

3.3.60 ssl_stop

Syntax	<i>wmiconfig --ssl_stop <server/client> <ssl index></i>	
Description	Stop SSL as either server or client.	
Parameters	<i><server/client></i>	Stop SSL server or client.
	<i><ssl index></i>	SSL index. The return value of ssl_start

3.3.61 ssl_config

Syntax	<i>wmiconfig --ssl_config <server/client> <ssl index> <protocol <protocol name>> [time <0 1>] [alert <0 1>] [domain <0/<name>] [cipher <cipher>]</i>	
Description	Configure SSL server or client.	
Parameters	<i><ssl index></i>	SSL index
	<i><protocol></i>	Protocol options: SSL3, TLS1.0, TLS1.1, TLS1.2, DTLS1.2
	<i><time></i>	Disable/enable certificate time validation. (Optional)
	<i><domain></i>	Disable/enable validation of peer's domain name against certificate common name (optional)
	<i><alert></i>	Disable/enable sending SSL alert when certificate validation fails. (Optional)
	<i><cipher></i>	Select cipher suite to use. Can be repeated 8 times. (Optional)
Example	<pre>wmiconfig --ssl_start client wmiconfig --ssl_config client 1 protocol DTLS1.2 time 1 alert 1 benchtx 192.168.0.104 7000 dtls 1350 1 3 0 1</pre>	

3.3.62 ssl_add_cert

Syntax	<i>wmiconfig --ssl_add_cert <server/client> <ssl index> <certificate/calist> [<name>]</i>	
Description	Add a certificate or CA list to SSL server or client.	
Parameters	<i><ssl index></i>	SSL index
	<i><name></i>	Name of the file to be loaded from flash

3.3.63 ssl_store_cert

Syntax	<i>wmiconfig --ssl_store_cert <name></i>	
Description	Store a certificate or CA list to the flash.	
Parameters	<i><name></i>	Name of the file to be stored to the flash.

3.3.64 ssl_delete_cert

Syntax	<i>wmiconfig --ssl_delete_cert <name></i>	
Description	Delete a certificate or CA list by name from flash.	
Parameters	<i><name></i>	Name of the file to be deleted from the flash.

3.3.65 ssl_list_cert

Syntax	<i>wmiconfig --ssl_list_cert</i>
Description	List the names of the certificates and CA lists stored in the flash.
Parameters	None

3.3.66 wdt

Syntax	<i>wmiconfig --wdt <0 1> <timeout></i>	
Description	Enable/Disable the watchdog timer	
Parameters	<i><0 1></i>	0 = Disable the watchdog 1 = Enable the watchdog
	<i><timeout></i>	Watchdog timeout in seconds

3.3.67 device

Syntax	<i>wmiconfig --device <device ID></i>	
Description	Select the device ID to which the subsequent <i>wmiconfig</i> commands take effect on. This command is used only when P2P concurrent mode of operation is enabled. Currently only two concurrent devices are supported, namely device 0 and device 1. Device 0 supports P2P/AP mode of operation whereas device 1 is intended for conventional STA (station) mode of operation.	
Parameters	<i><device ID></i>	0 = Select device 0 1 = Select device 1

3.3.68 ota_upgrade

Syntax	<i>wmiconfig --ota_upgrade <serverip> <filename> <mode> <preserve_last> <protocol></i>	
Description	Initialize OTA Service.	
Parameters	<i><serverip></i>	IP address of OTA Server
	<i><filename></i>	Filename of the file for OTA
	<i><mode></i>	0 = Firmware Download via OTA 1 = Host Download via OTA
	<i><preserve_last></i>	0 = Allow overwrite of the current partition by OTA. 1 = Do not allow overwrite of the current partition by OTA The <i>preserve_last</i> parameter is mainly used when only one alternative partition is available (which happens to be the current one). Unsuccessfully updating that partition results in falling back to the default partition that could be old.
	<i><protocol></i>	0 = TFTP. Only TFTP mode is supported.

3.3.69 ota_read

Syntax	<i>wmiconfig --ota_read <offset> <size></i>	
Description	Read size bytes from the offset to the OTA area	
Parameters	<i><offset></i>	Offset into the OTA area

	<size>	Number of bytes to read from the OTA area. Max allowed is 1024 byte chunks.
--	--------	--

3.3.70 ota_done

Syntax	<i>wmiconfig --ota_done</i>
Description	Complete and terminate OTA Session.
Parameters	None

3.3.71 i2c

Syntax	<i>wmiconfig --i2c master <read/write> <dev> <addr> <val></i> <i>wmiconfig --i2c slave <start/stop></i>	
Description	For I ² C master, perform read or write operation for the I ² C device. For I ² C slave, start or stop to receive the data in different mode.	
Parameters	<i>dev</i>	Device type
	<i>addr</i>	The address I ² C operating on
	<i>val</i>	For write operation only. The input value to be written into the I ² C device-dedicated address

3.3.72 pwm

Syntax	<i>pwm <module> <start/stop> <port group></i> <i>pwm <module> config <freq> <duty cycle> <phase> <port id> <src clk></i>	
Description	PWM operation, start or stop, config the PWM parameters	
Parameters	<i>module</i>	Module type: 0: pwm port module 1: sdm module (not recommended)
	<i>port_id</i>	0-7, totally 8 pwm channels.
	<i>port group</i>	0x1-0xff Every bit represents one channel: 0x1: port_id 0 0x2: port_id 1 0x3: port_id 0 and port_id 1
	<i>freq</i>	Module 0: 3-132000000 (0.03Hz-1.32MHz) Module 1: 0-200
	<i>start/stop</i>	Enable/disable port<port id> pwm output
	<i>duty cycle</i>	Module 0: 0-10000 Module 1: 0-255
	<i>phase</i>	Module 0: 0-10000
	<i>src clk</i>	0: CPU_CLK 1: REF_CLK 2: LPO_CLK Any other value: Defaults to 0

3.3.73 i2s

Syntax	<i>i2s <speaking recorder><config><start stop></i>	
Description	I ² S playback start stop or I ² S recorder start stop.	
Parameters	<i><speaking recorder></i>	speaking: I ² S playback; recorder: I ² S recorder
	<i><start stop ></i>	I ² S start or stop
	<i><config></i>	Choose I ² S port 0 or 1, master/slave mode

3.3.74 adc

Syntax	<i>adc light/pot/motion <freq> <accuracy> <dataCnt></i>	
Description	According to configuration, start ADC conversion and send the light/pot/motion sensor data to TCP server.	
Parameters	<i><freq></i>	Sample rate, 0-7. 0 = 31.25 KHz 1 = 1 MHz 2 = 500 KHz 3 = 250 KHz 4 = 100 KHz 5 = 50 KHz 6 = 25 KHz 7 = 10 KHz
	<i><accuracy></i>	Accuracy, 6-12. Larger number indicates higher accuracy.
	<i><dataCnt></i>	Total data count that all the channels should complete. If the data count is 0, the ADC converts data until the application stops/aborts it.

3.3.75 uart

Syntax	<i>uart <uart0 uart1 uart2><rx rts><enable disable></i>	
Description	Enable or disable UART wakeup function.	
Parameters	<i><uart0 uart1 uart2></i>	UART port number
	<i><rx rts></i>	Wakeup method: break signal (rx) or UART flow control (rts)
	<i><enable disable></i>	Enable/disable UART wakeup function

3.3.76 setbaudrate

Syntax	<i>setbaudrate <uart0 uart1 uart2><value></i>	
Description	Set baud rate of current UART	
Parameters	<i><uart0 uart1 uart2></i>	Current UART port number
	<i><value></i>	Baud rate value, currently support: 4800, 9600, 19200, 38400, 115200

3.3.77 pingld

Syntax	<i>wmiconfig --pingId <id></i>	
Description	Set Ping ID of Ping packet	
Parameters	<i><id></i>	Ping ID value

3.3.78 ip_dns_timeout

Syntax	<i>wmiconfig --ip_dns_timeout <timeout></i>	
Description	Set DNS timeout value	
Parameters	<i><timeout></i>	DNS timeout value in seconds

3.3.79 mcastfilter

Syntax	<i>wmiconfig --mcastfilter <enable/disable></i>	
Description	Enable/disable multicast filter	
Parameters	<i><enable/disable></i>	Enable or disable

3.3.80 print

Syntax	<i>wmiconfig --print <enable/disable></i>	
Description	Enable/disable debug information inside firmware	
Parameters	<i><enable/disable></i>	Enable or disable

3.3.81 ota_ftp

Syntax	<i>wmiconfig --ota_ftp <ip> <port> <user name> <password> <filename> <flags> <partition_index></i>	
Description	OTA FTP upgrade service	
Parameters	<i><ip></i>	IP address of OTA FTP Server
	<i><port></i>	Port of OTA FTP Server
	<i><user name></i>	Username
	<i><password></i>	Password
	<i><filename></i>	Filename of the file for OTA
	<i><flags></i>	QCOM_OTA_TARGET_FIRMWARE_UPGRADE (0x01) <ul style="list-style-type: none"> MUST be set. QCOM_OTA_PRESERVE_LAST (0x02) <ul style="list-style-type: none"> Set: The OTA engine does not overwrite the last active target firmware partition. In a two-partition system, this bit does not work after the first successful upgrade because the last active partition is always the second partition (golden partition is never updated). QCOM_OTA_ERASING_RW_DSET (0x04) <ul style="list-style-type: none"> Set: Erase all RW data sets in the selected partition if it is also the active partition. Default behavior is to retain RW data sets.

	<code><partition index></code>	Index of the requested partition to be upgraded, or 0 to allow the OTA engine to automatically select the partition.
--	--------------------------------------	--

3.3.82 ota_http

Syntax	<code>wmiconfig --ota_http <ip> <port> <filename> <flags> <partition_index></code>	
Description	OTA HTTP upgrade service	
Parameters	<code><ip></code>	IP address of OTA HTTP Server
	<code><port></code>	Port of OTA HTTP Server
	<code><filename></code>	Filename of the file for OTA
	<code><flags></code>	QCOM_OTA_TARGET_FIRMWARE_UPGRADE (0x01) <ul style="list-style-type: none"> ▪ MUST be set. QCOM_OTA_PRESERVE_LAST (0x02) <ul style="list-style-type: none"> ▪ Set: The OTA engine does not overwrite the last active target firmware partition. In a two-partition system, this bit does not work after the first successful upgrade because the last active partition is always the second partition (golden partition is never updated). QCOM_OTA_ERASING_RW_DSET (0x04) <ul style="list-style-type: none"> ▪ Set: Erase all RW data sets in the selected partition if it is also the active partition. Default behavior is to retain RW data sets.
	<code><partition index></code>	Index of the requested partition to be upgraded, or 0 to allow the OTA engine to automatically select the partition.

3.3.83 ota_https

Syntax	<code>wmiconfig --ota_https <ip> <port> <filename> <flags> <partition_index></code>	
Description	OTA HTTPS upgrade service	
Parameters	<code><ip></code>	IP address of OTA HTTPS Server
	<code><port></code>	Port of OTA HTTPS Server <ul style="list-style-type: none"> ▪ Should be 443 or 46030
	<code><filename></code>	Filename of the file for OTA
	<code><flags></code>	QCOM_OTA_TARGET_FIRMWARE_UPGRADE (0x01) <ul style="list-style-type: none"> ▪ Must be set. QCOM_OTA_PRESERVE_LAST (0x02) <ul style="list-style-type: none"> ▪ Set: The OTA engine does not overwrite the last active target firmware partition. In a two-partition system, this bit does not work after the first successful upgrade because the last active partition is always the second partition (golden partition is never updated). QCOM_OTA_ERASING_RW_DSET (0x04) <ul style="list-style-type: none"> ▪ Set: Erase all RW data sets in the selected partition if it is also the active partition. Default behavior is to retain RW data sets.
	<code><partition index></code>	Index of the requested partition to be upgraded, or 0 to allow the OTA engine to automatically select the partition.

3.3.84 bmiss

Syntax	<i>wmiconfig --bmiss <bmiss_num></i>	
Description	Configure the threshold for WLAN STA to report beacon miss, in unit of number of beacons	
Parameters	<i><bmiss_num></i>	The number of beacons, which WMAC does not receive from current AP continuously.

3.3.85 appleie

Syntax	<i>wmiconfig --appleie <enable disable></i>	
Description	Enable/disable a sample Apple information element of management frames (beacon, probe req, probe resp, assoc req)	
Parameters	<i><enable disable></i>	enable: add a sample Apple IE to the end of management frames. disable: delete the sample Apple IE from the management frames.

3.3.86 ani

Syntax	<i>wmiconfig --ani <0 1></i>	
Description	Enable/disable ANI	
Parameters	<i><0 1></i>	1: Enable 0: Disable

3.3.87 partition_index

Syntax	<i>wmiconfig --partition_index</i>	
Description	Get firmware partition index from which system booted	
Parameters	None	

3.3.88 timer_socket

Syntax	<i>timer_socket --us_timer <echo send run> <1 0> <test_flag> <IP> <PORT> </i> <i>timer_socket --exit</i>	
Description	Start/Stop the 15.625 ms timer to send or echo UDP data	
Parameters	<i>< --us_timer ></i>	echo send run: ▪ send = send UDP data to peer ▪ echo = receive UDP data from peer and send back ▪ run = just start the timer without any UDP action
		1 0: ▪ 1 = period. The timer is triggered periodically ▪ 0 = oneshot. The timer is triggered only one time
		test flag: stop the while loop manually in socket in the demo

		IP: <ul style="list-style-type: none"> When working as sender: IP address of peer side When working as receiver: IP address of itself
		PORT: transmission port
	< --exit >	Stop the 15.625 ms timer

3.4 Benchmark component

The benchmark component can be used to run uplink and downlink TCP/UDP streams to analyze throughput performance. The throughput demo tool is used to control the remote end of the stream. The tool is run on a PC using a command line.

Before running the throughput tests, make sure that the remote PC is connected to the same network. Up to two simultaneous streams are supported. The benchmark component is enhanced to demonstrate the usage of RAW sockets to send/receive RAW IP packets and EAPOL frames.

3.4.1 Benchmark commands

This section discusses the benchmark commands supported by the throughput demo tool.

Table 3-2 Benchmark commands

Command	Description
benchtx	Run the transmit (Tx) traffic test
benchtx raw socket	Run the transmit (Tx) traffic test on raw socket
benchrx	Run the receive (Rx) traffic test
benchrx raw socket	Run the receive (Rx) traffic test on raw socket
benchquit	Quit Tx/Rx traffic test

3.4.1.1 benchtx

Command	Run the transmit traffic test	
Syntax	<i>benchtx <Rx IP> <port> <protocol> <size> <test mode> < packets time> <delay> <SSL index></i>	
Description	The remote end must be in Rx mode before this test is run. The test may be run for a specified number of packets or a specified time period, and can be terminated using the command benchquit .	
Parameters	<Rx IP>	IP address of the remote end (receiver); in the format [A.B.C.D]

	<port>	Listening port	
	<protocol>	tcp	TCP
		udp	UDP
		ssl	SSL
	<size>	Packet size in bytes	
	<test mode>	0	Time test (test run for a specified time period)
		1	Packet test (test run for specified number of packets)
	<packets / time>	Test mode = 0: Enter the test time in seconds. Test mode = 1: Enter the number of packets to transmit.	
	<delay>	Delay in ms between packets (bandwidth control). This delay is useful to prevent one stream from starving the other in cases where two streams run simultaneously.	
	<SSL index>	When running SSL, this parameter indicates the index that benchrx/benchtx uses. The max index of Tx or Rx is 4.	

3.4.1.2 benchtx raw socket

Syntax	<i>benchtx <Rx IP> <prot> <raw> <msg size> <test mode> <number of packets / time (sec)> <delay in ms> <local IP> [ip_hdr_inc]</i>		
Description	Run the transmit traffic test. Before running this test, make sure the remote end is set to Rx mode. The test can be run for a number of packets or a time period. To terminate the test, enter <i>benchquit</i> .		
Parameters	<Rx IP>	IP address of the remote end (receiver) in format of [A.B.C.D]	
	<prot>	For IP RAW, this field takes value from 0 to 255. Protocols like ICMP, TCP and UDP are not supported. For Non-IP raw, the value for this field is eapol.	
	<raw>	tcp Indicates that the socket type is RAW.	
	<size>	Packet size in bytes	
	<test mode>	0	Time test (test run for a specified time period)
		1	Packet test (test run for specified number of packets)
	<packets/time>	Test mode = 0: Enter the test time in seconds. Test mode = 1: Enter the number of packets to transmit.	
	<delay>	Delay in ms between packets (bandwidth control). The delay is used to prevent one stream starving the other in cases where two streams run simultaneously.	
	<local IP>	IP address of the interface on which the RAW IP packets or EAPOL frames are to be transmitted.	
	[ip_hdr_inc]	This is an optional field. When configured, application will receive IP packets with IP Header.	

3.4.1.3 benchrx

Command	Run the receive traffic test		
Syntax	<i>benchrx <protocol> <port> [multicast IP] [Local IP] <SSL index></i>		
Description	The command sets the client in receive mode waiting for data. After the setting, the remote PC can be initiated to transmit data. The test can be terminated using the command <i>benchquit</i> .		
		tcp	TCP receiver

Parameters	<protocol>	udp	UDP receiver
		ssl	SSL
	<port>	Receiver port	
	[multicast IP]	IP address of multicast group to join (optional)	
	[local IP]	IP address of interface (optional)	
	<SSL index>	When running SSL, this parameter indicates the index benchrx/benchtx uses. The max index of Tx or Rx is 4.	

3.4.1.4 benchrx raw socket

Syntax	<i>benchrx raw <prot> <multicast address local address> [ip_hdr_inc]</i>		
Description	Run the receive traffic test. The command puts the client in receive mode, waiting for data. The remote PC tool shall be initiated to transmit data. To terminate the test, enter <i>benchquit</i> .		
Parameters	<prot>	For IP RAW, this field takes values from 0 to 255. Protocols like ICMP, TCP, and UDP are not supported. For Non-IP RAW sockets, use the string eapol for this field	
	<multicast address local address>	IP address of multicast group to join or IP address of interface for unicast	
	<ipdr_inc>	(Optional) This option enables the application to receive packet with Header	
Example	benchrx raw 88 224.0.1.1		

3.4.1.5 benchquit

Command	Quit traffic test
Syntax	<i>benchquit</i>
Description	Quit all Tx/Rx traffic test sessions.
Parameters	None

3.4.2 Multi-stream test

This section describes the best practices for running the bidirectional traffic tests using the throughput demo application from PC. The following rules must be observed during the test:

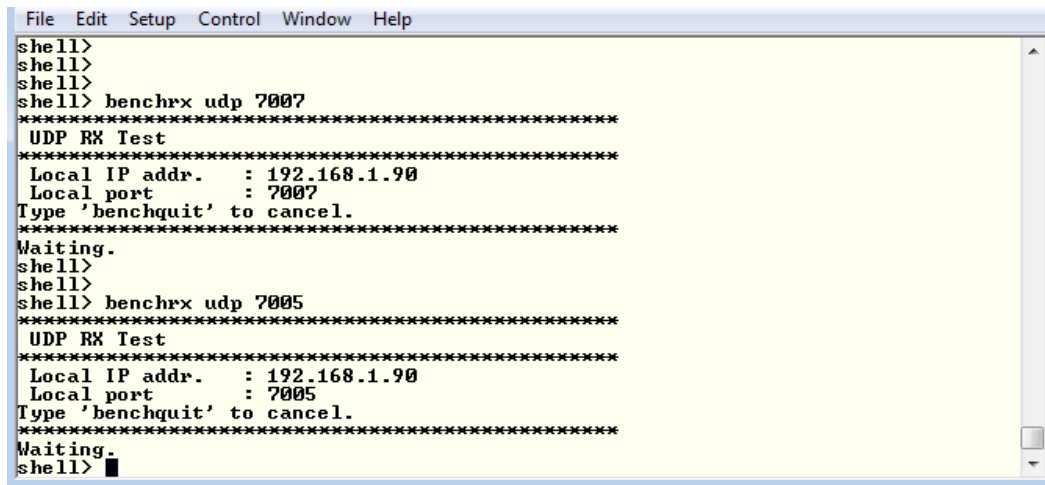
- Always set up the receiver before initiating any traffic stream.
- UDP stream can overpower TCP during TCP+UDP test. To allow TCP stream, a substantial delay is required between UDP packets.
- When running uplink and downlink stream simultaneously, avoid using the same port numbers for the two streams because the UDP transmitter uses the Rx port for feedback purposes.

3.4.2.1 Host Rx + host Rx (two simultaneous downlink streams)

This section is valid for UDP + UDP, TCP + TCP, and UDP + TCP tests.

1. Enter the first receive command. The host prints a *waiting...* prompt.

2. Enter the second receive command. Do not start any traffic.
3. Start two Tx streams from the two instances of the PC application.



```

File Edit Setup Control Window Help
shell>
shell>
shell>
shell> benchrx udp 7007
*****
UDP RX Test
*****
Local IP addr. : 192.168.1.90
Local port    : 7007
Type 'benchquit' to cancel.
*****
Waiting.
shell>
shell>
shell> benchrx udp 7005
*****
UDP RX Test
*****
Local IP addr. : 192.168.1.90
Local port    : 7005
Type 'benchquit' to cancel.
*****
Waiting.
shell>

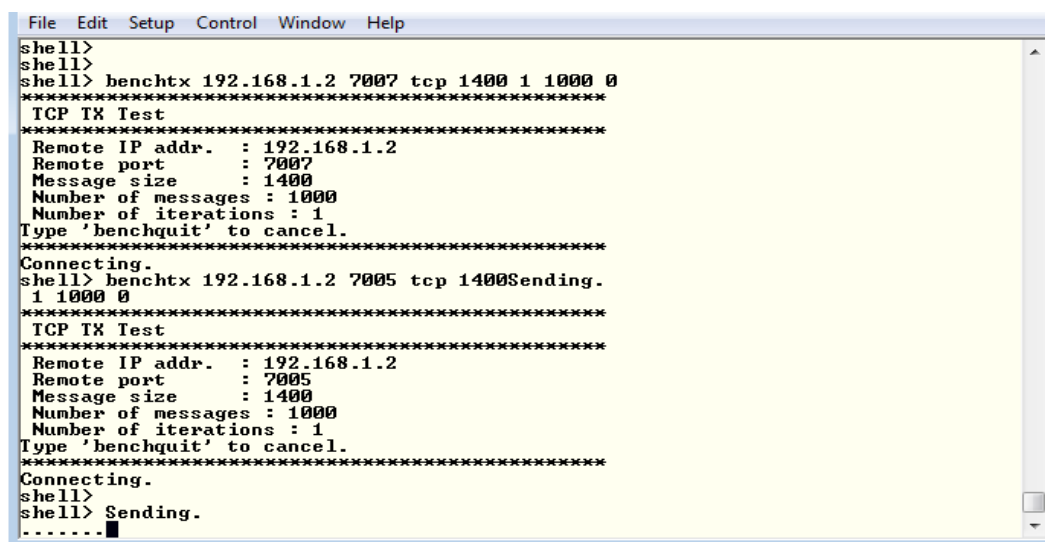
```

3.4.2.2 Host Tx + host Tx (two simultaneous uplink streams)

This section is valid for UDP + UDP, TCP + TCP, and UDP + TCP tests.

1. Set the two instances of the throughput demo application to receive mode.
2. Start first transmit stream.
3. Start the second transmit stream.

NOTE: The two commands can be pasted at once on some console terminals. If the console does not support it, the second command has to be typed manually.



```

File Edit Setup Control Window Help
shell>
shell>
shell> benchtx 192.168.1.2 7007 tcp 1400 1 1000 0
*****
TCP TX Test
*****
Remote IP addr. : 192.168.1.2
Remote port    : 7007
Message size   : 1400
Number of messages : 1000
Number of iterations : 1
Type 'benchquit' to cancel.
*****
Connecting.
shell> benchtx 192.168.1.2 7005 tcp 1400
1 1000 0
*****
TCP TX Test
*****
Remote IP addr. : 192.168.1.2
Remote port    : 7005
Message size   : 1400
Number of messages : 1000
Number of iterations : 1
Type 'benchquit' to cancel.
*****
Connecting.
shell>
shell> Sending.
.....

```

3.4.2.3 Host Tx + host Rx (simultaneous uplink and downlink streams)

This section is valid for UDP + UDP, TCP + TCP, and UDP + TCP tests.

1. Enter the receive command (the host prints a *waiting...* prompt).
2. Start the first instance in receive mode. Do not start any traffic.
3. Enter the transmit command and start transmission.

The host must start transmission before the PC application starts transmission. This is due to the incoming data process taking higher priority over the shell, preventing user from typing the transmit command.

4. Start the transmission from the second instance.

```

File Edit Setup Control Window Help
shell>
shell>
shell>
shell>
shell>
shell> benchrx udp 7007
*****
UDP RX Test
*****
Local IP addr. : 192.168.1.90
Local port : 7007
Type 'benchquit' to cancel.
*****
Waiting.
shell>
shell> benchtx 192.168.1.2 7005 udp 1400 1 100000 0
*****
UDP TX Test
*****
Remote IP addr. : 192.168.1.2
Remote port : 7005
Message size : 1400
Number of messages : 100000
Number of iterations : 1
Type 'benchquit' to cancel.
*****
Connecting.
Sending.
shell> Receiving from 192.168.1.2:54206

```

3.4.3 Throughput command line tool

The throughput tool includes a command line version to provide similar functionality with the GUI version. Multiple versions of the tool can be run concurrently.

The command line versions for transmit and receive are:

Transmit command:

```
ath_console.exe tx <Remote IP Address> <Remote Port/Protocol> <Protocol
[tcp|udp|raw]> <size> <mode> <arg> <delay> <v4|v6> <traffic class
[Optional]> <Local IP for RAW Tx>
```

Receive command:

```
ath_console.exe rx <Local IP Address> <Local Port> <Protocol [tcp|udp|raw]>
<v4|v6> <Multicast IP address [optional]>
```

Where:

size	Size of transmit packet in bytes		
mode	Test mode	0	Transmit for the specified time, in seconds
		1	Transmit for the specified number of packets
arg	If test mode = 0	<transmission time in seconds>	

	If test mode = 1	<number of packets>
delay	Time between packets, in milliseconds	
v4 v6	The user must specify either v4 or v6 for each test to select the correct version of IP protocol. Note that IPv6 is dependent on router and PC support.	
traffic class	Optional parameter This parameter is only available in the command line version of the tool. It is not available on the MQX platform.	
	BE	Best effort
	BG	Background
	VI	Video
	VO	Voice

The following rules must be kept in mind when using the command line version throughput tool:

- Do not use same value for the remote and the local port when Tx and Rx tests are run simultaneously. The UDP transmitter uses the specified port for the feedback mechanism.
- After the test is completed, press Ctrl + C to exit.
- The traffic class option is not available in the GUI version.
- The traffic class option is not available on the MQX platform because the RTCS stack does not support 802.1p QoS classes.

3.5 HTTP Server

Hyper-text transport protocol (HTTP) consists of the **HTTP Client** (web browser) that receives and views web pages, and the **HTTP Server** that stores, organizes, and transfers the web pages. For the request/response protocol, the browser requests a web page from the server and the server responds with the web page contents. On QCA4010/QCA4012, the HTTP Server supports both GET and POST methods.

3.5.1 Dynamic HTML

Standard HTML is static and does not support dynamic content. Dynamic HTML pages can be generated by including SSI directives that are replaced with the dynamic data. The SSI directives allow dynamic data to be inserted in to HTML files without programming. The QCA4010/QCA4012 supports the “echo” SSI directive element.

For example:

```
<p> Authen is <!--#echo var="auth" --></p>
```

auth is replaced with the configured value (for example, wpa/wpa2). HTTP Server generates dynamic HTML pages by replacing the tag with the data and transmits them to the browser when processing the GET request from the browser.

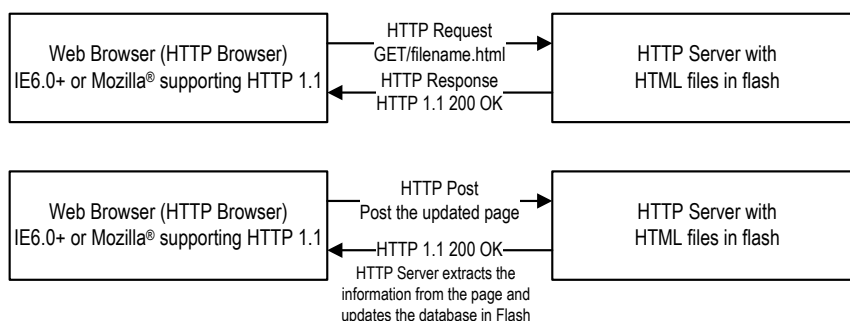


Figure 3-2 HTTP GET method

3.5.2 HTTP server GET and POST method test

Description	This test demonstrates the GET and POST methods of the HTTP server.	
Procedure	<ol style="list-style-type: none"> 1. Create the HTML files. Refer to the website below to create an HTML file. http://www.w3schools.com/html/ Make sure to include the SSI directive echo for every configurable element in the HTML page to get a dynamic page for a given static page. See Example HTML file. 2. Create a file filelist that contains the complete pathname of each HTML file, one per line. 3. Use target/tool/qonstruct.sh to build the flash image. <ol style="list-style-type: none"> a. Look for HTMLTOFLASH_LIST in the tunable_input.txt file and follow the instructions to point to the filelist. b. Run qonstruct.sh to create the raw_flash image. 4. Use gdb.sdk_flash to flash the raw_flash image to the Wi-Fi module. 5. Configure the reference AP to open security. 6. Start the HTTP Server on the development kit. 7. Associate the development kit and the station with the same reference AP. 8. Open a web browser on the reference station and enter the URL: <code>http://<DUT_IP_ADDRESS>/filename.html</code> 9. Check whether the requested HTML is displayed on the browser. 	
wmi commands	HTTP Server can be enabled by invoking the following commands. The reference AP is set to open security.	
	Associate the development kit with the reference AP	<code>wmiconfig --connect <SSID></code>
	Start HTTP Server on the development kit	<code>wmiconfig --ip_http_server start</code>

3.5.3 Example HTML file

```

-----
<html>
<head><title>IOE Kingfisher</title></head>
<body>
<form action="index.cgi" enctype="application/x-www-form-urlencoded" method="post"
name="cred">
<h1>IOE QCA4010 Simple Connect v2</h1>
<p>SSID <input maxlength="32" name="ssid" size="32" type="text" /></p>
<p> SSID is <!--#echo var="ssid" --></p>

```

```

<p>Authentication <input name="auth" type="radio" value="open"/>Open<input name="auth"
type="radio" value="wpa"/>WPA<input name="auth" type="radio" value="wpa2"/>WPA2</p>
<p> Authen is <!--#echo var="auth" --></p>
<p>Encryption <input name="sec" type="radio" value="none"/>None<input name="sec"
type="radio" value="tkip"/>TKIP<input name="sec" type="radio" value="AES"/>CCMP</p>
<p> Encryption is <!--#echo var="sec" --></p>
<p>Passphrase <input maxlength="32" name="pass" size="32" type="password" /></p>
<p> Passphrase is <!--#echo var="pass" --></p>
<p><input name="Submit" type="submit" value="Submit"> <input name="Reset" type="reset"
value="Reset"></p>
</form>
</body>
-----

```

1. CGI is specified in HTML files that contain dynamically updated parameters, in another word, CGI is not specified in HTML files containing static elements only. The web server stores the updated values in the database for the host to query. When CGI is specified in an HTML file, browser generates an HTTP request with a **<filename>.cgi** form action and then the HTTP Server invokes the function to parse the HTML files. The HTML form action (server-side form handler) **<filename>.cgi** is a virtual file. For example, if the HTML file name is **index.html**, the action is **index.cgi**.
2. Files must not use the same filenames (without extensions). For example, index.html and index.htm are not supported.
3. No file can have the extension **cgi**.
4. The maximum length of the SSI name is 32 characters.

3.6 HTTP Client

NOTE: HTTP Client is deprecated. For nonlegacy system, use multi-session HTTP Client (see section 3.7).

The HTTP Client uses HTTP to connect to a web server to transfer data. The most well-known HTTP clients are web browsers. In QCA4010/QCA4012, PUT, PATCH, GET and POST methods are supported and the application can act on the received data. The demo application only prints the data. The HTTP Client supports HTML pages under 1400 bytes. Only one operation is permitted at a time; that is, unless a request processing is complete, no new request can be issued.

3.6.1 HTTP PUT, PATCH, GET and POST methods test

Description	This test demonstrates the PUT, PATCH, GET and POST methods of the HTTP Client.	
WMI commands	HTTP Client can be enabled by invoking the following commands.	
	Connect to any HTTP server	<pre>wmiconfig --ip_http_client connect <ipaddress_or_domain> <port_num></pre> <p>Where:</p> <ul style="list-style-type: none"> ▪ <i>ipaddress</i> can be IPv4, IPv6, or a domain name for example www.qualcomm.com ▪ <i>port_num</i> is optional. If no port is provided, the client connects to port 80. <p>PUT/PATCH/GET/POST requests can be issued in sequence once the connection is established.</p>

	Request a page (print the requested page or error message)	PUT: <code>wmiconfig --ip_http_client put <url></code> PATCH: <code>wmiconfig --ip_http_client patch <url></code> GET: <code>wmiconfig --ip_http_client get <url></code> POST: <code>wmiconfig --ip_http_client post <url></code> Before posting, the name value pair can be updated using the command: <code>wmiconfig --ip_http_client query <name> <value></code>
--	--	---

3.6.2 Setup example

In this example two devices are configured, one running the HTTP Server and the other acting as the Client.

1. Connect to the HTTP server and update the example HTML page.

```
wmiconfig --ip_http_client connect <ipaddress>
```

A success message displays.

2. Request the index page.

```
wmiconfig --ip_http_client get /index.html
```

The page displays at the console.

3. Type the following commands. Once the executions are complete, the POST request is sent to the server with these name value pairs.

```
wmiconfig --ip_http_client query ssid dlink
wmiconfig --ip_http_client query auth open
wmiconfig --ip_http_client query sec none
wmiconfig --ip_http_client query pass 1234567890
wmiconfig --ip_http_client post /index.cgi
```

4. Request the same HTML page again and check whether the page is updated.

3.7 Multi-session HTTP Client

Multi-session HTTP client can address the old http client functions in entirety (see section [3.6](#)). At most, four clients can be created. Each client can set its own request timeout and callback function.

3.7.1 Multi-session HTTP client test methods on CLI

Description	This test demonstrates of the multi-session HTTP Client.
WMI	HTTP Client can be enabled by invoking the following commands.

commands	Connect to any HTTP server	<pre>wmiconfig --ip_httpclient2_connect <server> <port> <timeout> [<ssl_ctx_index>] [<callback_enable>]</pre> <p>Where:</p> <ul style="list-style-type: none"> server can be IPv4, IPv6, or a domain name with or without http(s):// for example www.qualcomm.com port is the http server port on listening timeout is the http client request timeout value ssl_ctx_index is the ssl index callback_enable select whether enable callback mode
	Disconnect from http server	<pre>wmiconfig --ip_httpclient2_disconnect <client_num></pre> <ul style="list-style-type: none"> client_num is the index of created client.
	GET request	<pre>wmiconfig --ip_httpclient2_get <client_num> <url></pre> <ul style="list-style-type: none"> client_num is the index of created client. url is the url to access
	PUT request	<pre>wmiconfig --ip_httpclient2_put <client_num> <url></pre> <ul style="list-style-type: none"> client_num is the index of created client. url is the url to access
	POST request	<pre>wmiconfig --ip_httpclient2_post <client_num> <url></pre> <ul style="list-style-type: none"> client_num is the index of created client. url is the url to access
	PATCH request	<pre>wmiconfig --ip_httpclient2_patch <client_num> <url></pre> <ul style="list-style-type: none"> client_num is the index of created client. url is the url to access
	Set body	<pre>wmiconfig --ip_httpclient2_setbody <client_num></pre> <ul style="list-style-type: none"> client_num is the index of created client.
	Add header	<pre>wmiconfig --ip_httpclient2_addheader <client_num> <hdr_name> <hdr_value></pre> <ul style="list-style-type: none"> client_num is the index of created client. Hdr_name is the name of header to be set. Hdr_value is the value to be set to header.
	Clear header	<pre>wmiconfig --ip_httpclient2_clearheader <client_num></pre> <ul style="list-style-type: none"> client_num is the index of created client.
	Set parameter	<pre>wmiconfig --ip_httpclient2_setparam <client_num> <key> <value></pre> <ul style="list-style-type: none"> client_num is the index of created client. Key is the name of key to be set. Value is the value to be set to key.

3.7.2 Multi-session HTTP client setup example

This is the example that two HTTP client sessions run at the same time.

Test Device

- a DUT

- an HTTPS server PC (IP addr: 192.168.1.10)
- an AP

Test preparation

1. Connect DUT to AP and get the IP address.
2. Connect the HTTPS Server PC to AP and get the IP address.
3. Start the HTTPS Server.

HTTPS client CLI command

Start session 1:

```
Step 1: wmiconfig --ssl_start client
//SSL client started : Index 1
Step 2: wmiconfig --ip_httpclient2_connect https://192.168.1.10 443 8000
1 callback_enable
//http client connect Success <client num> = 1
Step 3: wmiconfig --ip_httpclient2_get 1 /index.html
//print the content of index.html.
```

Start session 2:

```
Step 4: wmiconfig --ssl_start client
//SSL client started: Index 2
Step 5: wmiconfig --ip_httpclient2_connect https://192.168.1.10 443 8000
2 callback_enable
//http client connect Success <client num> = 2
Step 6: wmiconfig --ip_httpclient2_get 2 /index.html
//print the content of index.html.
```

Disconnect:

```
Step 7: wmiconfig --ip_httpclient2_disconnect 1
Step 8: wmiconfig --ip_httpclient2_disconnect 2
```

NOTE: PUT, POST and PATCH are similar with the GET method.

3.8 Simple DNS Server

DNS Server translates domain names to IP addresses. Each domain is managed by a server. DNS Server associates the TCP/IP address assigned by DHCP server to a client with the HOST name advertised by the DHCP Client. The DHCP Server and DNS Server interact to create the database. Simple DNS Server is enabled only in Soft AP mode.

3.8.1 Simple DNS Server test (Soft AP mode)

Description	This test demonstrates a DNS Server association to a TCP/IP address. The TCP/IP address is assigned by the DHCP server to a client. The HOST name is advertised by the DHCP client as part of DHCP exchange.
--------------------	--

Procedure	<ol style="list-style-type: none"> 1. Configure the development kit to Soft AP mode. 2. On the Soft AP, configure the domain name for the DNS server to manage. 3. Configure the HOST name on the clients that will associate to the Soft AP. The clients can be the development kit or Windows PC. The development kit clients must be configured with a unique MAC address. 4. On the development kit client, configure the HOST name using the WMI command wmiconfig --iphostname. Then acquire an IP address using wmiconfig --ipdhcp. Once the stations are associated with the Soft AP and have acquired IP addresses, they can communicate with each other using the HOST names. 5. When STA-A pings STA-B using the HOST name, STA-A sends a query to the Soft AP to resolve the STA-B HOST name. 6. After STA-A learns the address of STA-B, it uses that address for subsequent interactions. 	
WMI commands	The following WMI commands can be used to demonstrate the use case of DNS server.	
	1. Configure Soft AP in open security.	<pre>wmiconfig --mode ap wmiconfig --channel 6 wmiconfig --connect ioeap wmiconfig --ip_dns_local_domain qca.com</pre>
	2. Configure the HOST name before acquiring IP address (in this example STA-A is the development kit).	<pre>wmiconfig --iphostname QCA4010 wmiconfig --ipdhcp</pre> <ul style="list-style-type: none"> ▪ Use ipconfig command to learn the IP address and HOST name configured on STA-A. ▪ The hostname and IP address acquired are displayed.
	<ol style="list-style-type: none"> 3. Repeat step 2 if STA-B is the development kit. 4. From STA-A, ping STA-B using the host name. STA-A sends a DNS query to the Soft AP to learn the IP address of STA-B. <p>The application on STA-A can now interact with the application on STA-B through the Soft AP by hostname. The API custom_ip_resolve_hostname can also be used to learn the IP address of the peer by domain name.</p>	

3.9 DNS Client

DNS Client resolves and caches domain name system (DNS) domain names. When the DNS Client receives a request to resolve a DNS name that does not exist in its cache, it queries a configured DNS Server for the IP address. When the DNS Client receives the requested address, it stores the name and address in its cache for future requests. DNS Client is enabled by default.

DHCP (if enabled) can configure one of the DNS servers. Up to three DNS servers can be configured manually. The throughput demo application has an enhanced ping command to work with the DNS client.

WMI commands can be used to add and delete the DNS server, and to resolve hostname for IPv4 and IPv6.

Description	This test demonstrates the usages of DNS Client.	
Procedure	DNS Client can be invoked using the commands:	
	Add DNS Server address.	<code>wmiconfig --ip_dns_server_addr <ipaddr></code>

	When DNS Server is added, the DNS Client can resolve domain names using the commands:	<pre>wmiconfig --ip_resolve_hostname [<host_name> <domain_type>] wmiconfig --ip_gethostbyname [<host_name>] wmiconfig --ip_gethostbyname2 [<host_name> <domain_type>]</pre>
	Directly ping the host.	<pre>ping <host_name></pre>

The **<domain_type>** parameter specifies the IP version of the requested address.

- **domain_type = AF_INET:** The DNS Client tries to get an IPv4 address for a given name by sending DNS Query of type DNS_TYPE_IPADDR.
- **domain_type = AF_INET6:** The DNS Client tries to get an IPv6 address for a given name by sending DNS Query of type DNS_TYPE_AAAA.

3.10 OTA firmware upgrade

The Over-The-Air (OTA) firmware upgrade feature allows upgrading firmware over WLAN connection. This feature is available on the development kit.

Multiple approaches can be adopted to download firmware from the OTA server to the Wi-Fi module. In release 4.0 only TFTP protocol-based communication is supported for OTA firmware upgrade. More approaches will be supported in future releases.

This section describes the TFTP-based OTA firmware update procedures on the development kit.

3.10.1 TFTP OTA firmware upgrade methods

OTA firmware upgrade is triggered by either of the following methods:

- Run OTA upgrade command from shell.
- Run OTA upgrade application on remote client connected to the Wi-Fi module over WLAN.

After the firmware upgrade is triggered:

1. The OTA firmware upgrade module creates a TFTP client task to download firmware from the OTA server.
2. Partition selection and image checksum validation for the downloaded image are internally performed by the firmware.
3. A wmiconfig read API is provided to the application for authenticating the downloaded image.
4. Once the application validates the downloaded image, the application must indicate to the firmware using `wmiconfig ota_done` command.
5. Reboot the development kit.

During the reboot process, the nvram initialization module checks for the latest firmware and tries loading the firmware. If successful, the QCA4010/QCA4012 runs from the latest image from the flash; otherwise the QCA4010/QCA4012 runs Image 0.

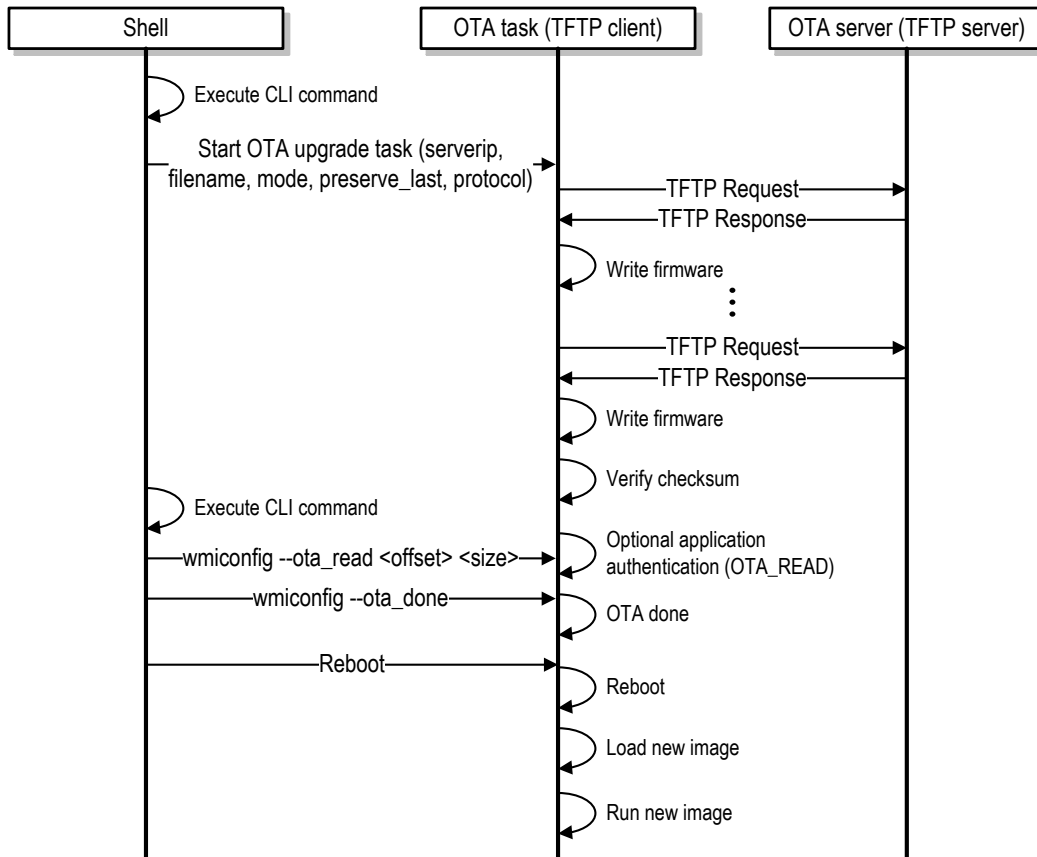


Figure 3-3 OTA firmware upgrade process

3.10.1.1 Trigger OTA firmware upgrade from shell

Procedure	<ol style="list-style-type: none"> 1. Upload the image file to the OTA server and start the server. 2. Connect a PC to the development kit via console port and connect the development kit to the OTA server over WLAN. Make sure the IP addresses are within the same subnet. 3. Run command wmiconfig --ota_upgrade from the Wi-Fi module shell to trigger firmware upgrade. 4. Run command wmiconfig --ota_done from the Wi-Fi module shell to close the OTA session. 5. Reboot Wi-Fi module and check the firmware version of the running image. 	
Syntax	wmiconfig --ota_upgrade <OTA-server-ip> <file-name> <mode> <preserve_last> <protocol>	
Parameters	<OTA-server-ip>	OTA server IP address
	<file-name>	File name
	<mode>	0 = Firmware/hostless OTA download 1 = Host OTA download
	<preserve_last>	0 = Allow overwriting the current partition for OTA download. 1 = Do not allow overwriting the current partition for OTA download. <mode=1> and <preserve_last=1> are not supported.
	<protocol>	Protocol type 0 = TFTP. Currently only TFTP is supported.

3.10.1.2 Read OTA image over console connection

Procedure	After OTA image is downloaded successfully, the application/host can use the OTA read API to authenticate or validate the image. Run <code>wmiconfig --ota_read</code> from the Wi-Fi module shell to read the OTA image. The entire length of the downloaded image can be read as reported by the OTA service.	
Syntax	<code>wmiconfig --ota_read <offset> <size></code>	
Parameters	<code><offset></code>	Offset to the OTA image
	<code><size></code>	Size of bytes to read from the OTA image offset. Up to 1k chunks of image can be read from the OTA image via the OTA read API.

3.10.1.3 Close OTA session

Procedure	After OTA image is authenticated or validated, run <code>wmiconfig --ota_done</code> to close the OTA session. After that, the host/application can reboot from the new image.	
Syntax	<code>wmiconfig --ota_done</code>	
Parameters	None	

3.10.1.4 OTA status codes

Following are the status codes returned by the OTA service.

```
QCOM_OTA_OK = 0,
QCOM_OTA_COMPLETED=1,

QCOM_OTA_ERR_UNKOWN_MSG = 1000,
QCOM_OTA_ERR_IMAGE_NOT_FOUND = 1001,
QCOM_OTA_ERR_IMAGE_DOWNLOAD_FAIL = 1002,
QCOM_OTA_ERR_IMAGE_CHECKSUM_INCORRECT = 1003,
QCOM_OTA_ERR_SERVER_RSP_TIMEOUT = 1004,
QCOM_OTA_ERR_INVALID_FILENAME = 1005,
QCOM_OTA_ERR_UNSUPPORT_PROTOCOL = 1006,
QCOM_OTA_ERR_INVALID_PARTITION_INDEX = 1007,
QCOM_OTA_ERR_IMAGE_HDR_INCORRECT = 1008,
QCOM_OTA_ERR_INSUFFICIENT_MEMORY=1009,
QCOM_OTA_ERR_PRESERVE_LAST_FAILED=1010,
QCOM_OTA_ERR_NO_ACTIVE_OTA_SESSION=1011,
QCOM_OTA_ERR_INVALID_PARTITION_ACESS=1012,
QCOM_OTA_ERR_OTA_SESS_IN_PROGRESS=1013,
QCOM_OTA_ERR_FLASH_READ_TIMEOUT=1014,
QCOM_OTA_ERR_FLASH_ERASE_ERROR=1015,
QCOM_OTA_ERR_IMAGE_OVERFLOW=1016,
QCOM_OTA_ERR_IMAGE_UNDERFLOW=1017,
QCOM_OTA_ERR_WRITE_DATA_ERROR=1018,
```

3.11 Fail-Safe flash partitioning

The hostless software supports multiple partitions in the flash device. The Partition 0 (P0) is always followed by P1, P2 and so on depending on the size of the flash. P0 is designated as the fail-safe partition (also referred to as the “root” partition).

When system failure occurs and the watchdog timer expires, the watchdog triggers a reset, which is handled uniquely by the ROM, to force the QCA4010/QCA4012 chip to load from the root partition so that the chip comes up in a known stable state.

The fail-safe flash behavior should not be confused with the `wmiconfig --reset` command, which triggers a reset to the system and the chip boots up from the latest partition.

3.12 WPS demo application

3.12.1 Station mode

System setup

- Required devices
 - Development kit
 - AP with WPS support
- AP configuration
 - Enable WPA-PSK or WPA2-PSK and configure a passphrase.
 - Enable WPS support.
- RB02 configuration

Enter the following command at the Wi-Fi module shell to make it operate in station mode.

```
wmiconfig --mode station
```

3.12.1.1 WPS connection using PIN method

1. At the Wi-Fi module shell, enter the following command to set the PIN for the Wi-Fi module and start WPS connection.

```
wmiconfig --wps 1 pin 12345670
```

2. On the AP, enter the PIN code of the Wi-Fi module. Refer to the AP user manual for configuration steps.
3. Observe the connection process at the Wi-Fi module shell. The connection is usually established within 120 seconds. When the connection is established, the following lines are printed on the shell.

```
connected to <ssid-of-AP>, bssid =<bssid-of-AP>, channel = <Channel Num>, rssi = <RSSI value>
```

4. On the Wi-Fi module, configure a static IP address in the same subnet with the AP.


```
wmiconfig --ipstatic <IP Address> <Subnet Mask> <Default Gateway>
```
5. On the Wi-Fi module, ping the AP to verify the WPS connection with the AP.


```
ping <IP of AP>
```

3.12.1.2 WPS connection using PUSH method

1. At the Wi-Fi module shell, enter the following command to emulate the action of WPS PUSH button and start WPS connection.

```
wmiconfig --wps 1 push
```

2. On the AP, push the WPS button. Refer to the AP user manual for details.
3. Observe the connection process at the Wi-Fi module shell. The connection is usually established within 120 seconds. When the connection is established, the following lines are printed on the shell.

```
connected to <ssid-of-AP>, bssid =<bssid-of-AP>, channel = <Channel Num>, rssi = <RSSI value>
```

4. On the Wi-Fi module, configure a static IP address in the same subnet with the AP.

```
wmiconfig --ipstatic <IP Address> <Subnet Mask> <Default Gateway>
```

5. On the Wi-Fi module, ping the AP to verify the WPS connection with the AP.

```
ping <IP of AP>
```

3.12.2 Soft AP mode

System setup

- Required devices
 - Development kit
 - Wireless station with WPS support
- Wi-Fi module configuration

Enter the following commands at the shell:

```
wmiconfig --mode ap
```

```
wmiconfig --p 12345678
```

```
wmiconfig --connect ioe-kf
```

- Station configuration

Refer to the user manual of the wireless station.

3.12.2.1 WPS connection using PIN method

1. On the station, start WPS connection process and get the PIN code. Refer to the station user manual for configuration steps.

2. At the Wi-Fi module shell, enter the following command to input the PIN of the station.

```
wmiconfig --wps 1 pin <pin of station>
```

3. Observe the connection process at the station. The connection is usually established within 120 seconds.

4. On the station, configure a static IP address in the same subnet with the Wi-Fi module.

5. On the Wi-Fi module, ping the station to verify the WPS connection.

```
ping <IP of station>
```

3.12.2.2 WPS connection using PUSH method

1. At the Wi-Fi module shell, enter the following command to emulate the action of WPS PUSH button and start WPS connection.

```
wmiconfig --wps 1 push
```
2. On the station, start the WPS process. Refer to the AP user manual for details.
3. Observe the connection process at the station. The connection is usually established within 120 seconds.
4. On the station, configure a static IP address in the same subnet with the Wi-Fi module.
5. On the Wi-Fi module, ping the station to verify the WPS connection.

```
ping <IP of station>
```

3.13 SSL

Secure socket layer (SSL) is a standard security technology for establishing encrypted link between a server and a client – typically a webserver (website) and a browser or a mail server and a mail client. Once the client and server have decided to use TLS, they negotiate a stateful connection by using a handshaking procedure. During this handshake, the client and server agree on various parameters used to establish the connection's security:

1. The client sends the server the client's SSL version number, cipher settings, session-specific data, and other information that the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, session-specific data, and other information that the client needs to communicate with the server over SSL. The server also sends its own certificate, and if the client is requesting a server resource that requires client authentication, the server requests the client's certificate.
3. The client uses the information sent by the server to authenticate the server. For example, in the case of a web browser connecting to a web server, the browser checks whether the received certificate's subject name actually matches the name of the server being contacted, whether the issuer of the certificate is a trusted certificate authority, whether the certificate has expired, and, ideally, whether the certificate has been revoked. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client proceeds to the next step.
4. Using all data generated in the handshake thus far, the client (with the cooperation of the server, depending on the cipher in use) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in step 2), and then sends the encrypted pre-master secret to the server.
 - If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.
 - If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, and then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.

5. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity (that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection).
6. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
7. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
8. The SSL handshake is now complete and the session begins. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

On QCA4010/QCA4012, new hardware crypto is supported and works for SSL, secure boot and HTTPs. QCA4010/QCA4012 also supports multi-session for Bench SSL. The maximum sessions is 4 both for Rx and Tx.

About HWCrypto it supports the following algorithms:

- Encryption: Contains the encryption algorithms: AES128, AES256 and DES/3DES.
- Authentication: Contains the authentication algorithms: SHA1 and SHA256.

3.13.1 Certificate management demo

The following test procedure is an example to demonstrate how to download an RSA or an ECC certificate to the device. The reference of “foo” is a name example.

1. Generate an RSA device certificate using OpenSSL.
 - Option 1: Generate an RSA certificate.
 - i Generate a private key.
`openssl genrsa -des3 -out foo.key 1024`
 - ii Generate a self-signed certificate.
`openssl req -new -key foo.key -out foo.pem -x509 -days 365`
 - iii Remove the passphrase from the key.
`mv foo.key foo.key.withpass`
`openssl rsa -in foo.key.withpass -out foo.key`
 - Option 2: Generate an ECC certificate.
 - iv Generate a private key.
`openssl ecparam -name secp224r1 -genkey -noout -out foo.key`
 - v Generate a self-signed certificate.
`openssl req -new -key foo.key -out foo.pem -x509 -days 365`
2. Convert certificate to SharkSSL binary format.
 - a. Generate binary certificate file.
`SharkSSLParseCert foo.pem foo.key -b foo.cert.bin`

- b. Generate binary CA List file.
`SharkSSLParseCAList -b foo.calist.bin foo.pem`
3. Start the certificate server on a Linux PC.
 - a. Copy the **foo.cert.bin** and **foo.calist.bin** files to the **certcs** directory that also contains the **certcs** executable.
 - b. Start the certificate server.
`./certcs -s`
 - c. Test the certificate server from another PC.
`./certcs HOSTNAME/foo.calist.bin`
 HOSTNAME is the host name of the PC where the certificate server is started.
4. Download the certificate and CA list from the server and write them to the flash.
 - a. Download and flash the certificate.
`getcert foo.cert.bin IPADDR -s kf.crt`
 - b. Download and flash the CA list.
`getcert foo.calist.bin IPADDR -s kf.ca`
 IPPADDR is the IP address of the certificate server.
5. Use the certificate and the CA list stored in the flash.
 The **kf.crt** and **kf.ca** will be subsequently used with the other SSL **wmiconfig** commands.
 - Add certificate to the server SSL context.
`wmiconfig --ssl_add_cert server 1 certificate kf.crt`
 - Add CA list to the client SSL context.
`wmiconfig --ssl_add_cert client 1 calist kf.ca`
 - Delete the files.
`wmiconfig --ssl_delete_cert kf.ca`
`wmiconfig --ssl_delete_cert kf.crt`

Test prerequisites

The following settings are required for running the test examples.

1. Connect the device to an AP.
`wmiconfig --connect SSID`
2. Assign either a static IP address or a dynamic IP address using DHCP client. The latter is preferred.
`wmiconfig --ipdhcp`
3. Start the SNMP client.
`wmiconfig ip_snmp_client start`

Test example, SSL/TLS server

```
wmiconfig --ssl_start server
wmiconfig --ssl_add_cert server 1 certificate kf.crt
benchrx ssl 1443 1
```

Test example, SSL/TLS client

1. On the Linux PC side, run the following command:

```
ncat -l --ssl --ssl-key foo.key --ssl-cert foo.pem -k 1443
```
2. On the Wi-Fi module side, run the following commands:

```
wmiconfig --ssl_start client
wmiconfig --ssl_add_cert client 1 calist kf.ca
wmiconfig --ssl_config client 1 protocol TLS1.2 time 1
benchtx 10.1.33.100 1443 ssl 1400 1 100 0 1
```

3.13.2 Cipher Suites list

The following lists the supported Cipher Suites:

- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC14)
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC02C)
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xC024)
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC02B)
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xC023)
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xC00A)
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xC009)
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 (0xC02E)
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xC026)
- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 (0xC02D)
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xC025)
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xC005)
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xC004)
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xCC13)
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC030)
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC028)
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC02F)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC027)
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC014)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC013)
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 (0xC032)
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xC02A)
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 (0xC031)
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xC029)
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xC00F)

- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xC00E)
- TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009D)
- TLS_RSA_WITH_AES_256_CCM (0xC09D)
- TLS_RSA_WITH_AES_256_CCM_8 (0xC0A1)
- TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003D)
- TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
- TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009C)
- TLS_RSA_WITH_AES_128_CCM (0xC09C)
- TLS_RSA_WITH_AES_128_CCM_8 (0xC0A0)
- TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003C)
- TLS_RSA_WITH_AES_128_CBC_SHA (0x002F)

3.14 Data set

When using NVRAM (SPI serial flash) as the storage media for data sets, two areas of flash can be defined and used:

- Read Only flash data set area

Data sets in the Read Only flash data set area will not be altered nor deleted except during a firmware upgrade. The Read Only flash data set area consumes only as much flash space as required by the aggregate of all data set in that data set area plus a small amount of overhead.

- Read/Write flash data set area

The Read/Write flash data set area starts with nothing in it. Firmware may dynamically create, modify and delete data sets in this area. Each data set in the Read/Write flash data set area consumes a number of 4KB flash blocks. For example, a data set with 128B of data consumes 4KB of flash.

Start address and size of the data set area

For Read Only flash data set area, the start address is determined by the build-time tunable variable **RODATASET_START**, the size is determined by the amount of data placed into it.

For Read/Write Flash data set area, the start address and the maximum size are determined by the build-time tunable variables **RWDATASET_START** and **RWDATASET_MAXSIZE**.

Both Read Only and Read/Write flash data set area may be located in the software partition (in which case the contents may be lost or changed during firmware upgrade), or be placed into an independent partition which will not be modified during firmware upgrade. See **tunable_input.txt** for details on how to use these tunables to locate and size of the data set areas.

NOTE: If a firmware image built with different start address or size tunable is programmed to the flash during firmware upgrade, the old data could be lost.

During operation, firmware supports only a single Read Only flash data set area plus a single Read/Write flash data set area.

Presence of data set

The presence of data set can be used to check whether certain activity has taken place or not. In the case of flash, a read failure indicates that data set has not been created yet and needs to be initialized and updated. If an AORAM data set is found, it indicates that store/suspend has been performed before (the AORAM remained powered during the suspend period). System started on a cold boot does not have AORAM data set.

Using the data set APIs

1. Include the **qcom_dset.h** file in the application.
2. Two scenarios for using the data set APIs.
 - Data set ID does not exist in the QCA4010/QCA4012 media.
 - i Create data set ID on the indicated media (qcom_dset_create).
 - ii Access the data set ID (qcom_dset_write).
 - iii Commit the data set after it is written (qcom_dset_commit).
 - Data set ID exists in the QCA4010/QCA4012 media.
 - iv Open the data set ID (qcom_dset_open).
 - v Access the data set ID (qcom_dset_read).
 - vi Close the data set after use (qcom_dset_close).

If the data set is no longer needed, delete it (qcom_dset_delete).

3.15 Dynamic GPIO configuration

Dynamic GPIO configuration is a feature enabling user to save power as well as to reuse GPIO pins for multiple peripheral interfaces at runtime.

Power consumption is critical for IOE devices which requires system to be highly optimized. The GPIOs configured in different ways to communicate with external peripherals contribute part of the chip current consumption. On QCA4010/QCA4012, some of the commonly used peripheral interfaces includes SPI master (for serial flash access), UART, and I²C/I²S (for CODEC/Thermostat communication). Configure GPIO pins properly to actively use them for a particular purpose. When a GPIO pin is inactive it should be reconfigured so as to reduce power consumption.

The current GPIO framework enables users to configure the GPIOs statically in the tunable input text file and these configurations cannot be changed later in runtime. For example, if a pin is configured for I²C, it cannot be used for any other peripherals such as I²S even if the hardware supports multiple peripherals on a given GPIO pin. To remove this limitation, dynamic GPIO configuration with pin-based active configuration set is implemented in this release.

This section provides details of the current GPIO framework as well as the implementation and programming procedures for dynamic GPIO configuration.

3.15.1 GPIO customization

QCA4010/QCA4012 includes a total of 41 GPIO pins (0 to 40) that can be configured for various functionality depending on the user cases. Some of the commonly used peripherals are SPI

Master (for serial flash access), UART, and I²C/I²S (for CODEC/Thermostat communication). Configure GPIO pins properly to actively use them for a particular purpose. When a GPIO pin is inactive it should be reconfigured so as to reduce power consumption.

To simplify the GPIO configurations, Qualcomm Technologies allows user to provide GPIO configuration inputs through the **tunable_input.txt** file. This file contains multiple tunable parameters including the GPIOs.

- GPIO Active configuration data (for all GPIO pins)

This configuration applies when the pin is configured for any peripheral or to GPIO mode.

- GPIO Inactive configuration data (for all GPIO pins)

This configuration applies when the pin is not in use.

- Interface-to-GPIO mapping data

This configuration provides the peripheral ID for a given GPIO pin that corresponds to the active configuration mentioned above. Only one peripheral ID can be associated to the GPIO Active configuration for each GPIO pin in the tunable input file. Alternate configurations can be added at run time using the **qcom_gpio_add_alternate_configurations** API.

Table 3-3 GPIO configurations

Interface to GPIO mapping			Inactive set Configuration	Active set Configuration
S.No	Interface	GPIO Pin Number	GPIO Pin Configuration	GPIO Pin Configuration
1	SPI Flash	PIN# A	GPIO0	GPIO0
		PIN# B	GPIO1	GPIO1
		PIN# C	GPIO2	GPIO2
		PIN# D	GPIO3	GPIO3
2	I2C	PIN# I	.	.
		PIN# J	.	.
		PIN# K	.	.
		PIN# L	.	.
3	I2S	PIN# X	.	.
		PIN# Y	.	.
		PIN# Z	.	.
..	GPIO40	GPIO40

The supported GPIO peripheral IDs are listed as follows:

```
/* GPIO peripheral IDs supported by Qualcomm Technologies hardware*/
#define GPIO_PERIPHERAL_ID_NONE      0
#define GPIO_PERIPHERAL_ID_UART_0    1
#define GPIO_PERIPHERAL_ID_UART_1    2
#define GPIO_PERIPHERAL_ID_I2C       3
#define GPIO_PERIPHERAL_ID_I2S       4
#define GPIO_PERIPHERAL_ID_I2S_SLAVE 5
#define GPIO_PERIPHERAL_ID_SPI_MASTER 6
```

```

#define GPIO_PERIPHERAL_ID_SPI_SLAVE      7
#define GPIO_PERIPHERAL_ID_NOT_BONDED    8
#define GPIO_PERIPHERAL_ID_PWM_HW        0x9
#define GPIO_PERIPHERAL_ID_PWM_SW        0xA
#define GPIO_PERIPHERAL_ID_GPIO          0xB
#define GPIO_PERIPHERAL_ID_JTAG           0xC
#define GPIO_PERIPHERAL_ID_I2C_SLAVE      0xD
#define GPIO_PERIPHERAL_ID_QUAD_SPI_MASTER 0xE
#define GPIO_PERIPHERAL_ID_UART_2        0xF
#define GPIO_PERIPHERAL_ID_MAX

```

The GPIO Active configuration, GPIO Inactive configuration, and GPIO-peripheral mapping data are provided in **tunable_input.txt** which is taken by **qonstruct.sh** script to generate binary. Every data entry of GPIO Active and Inactive configuration is a 32-bit integer which encodes multiple information in bitmaps. [Table 3-4](#) provides the bitmap structure and the possible values.

Table 3-4 GPIO active/inactive configuration

Bit	Field	Value	Description
0	Source	0	WLAN_GPIO_OUT register
		1	Sigma Delta PWM Resource
1	Pull Override	0	No pull override
		1	Override to pull down
2	Pull Driver	0	Push/Pull driver
		1	Open drain driver
4:3	Pad Strength	00	Pad strength 4 mA
		01	Pad strength 8 mA
		10	Pad strength 12 mA
		11	Pad strength 24 mA
6:5	Pad Pull	00	No pull
		01	Pull up
		10	Pull down
		11	Reserved
9:7	Interrupt Config	000	Interrupt disabled
		001	Interrupt on rising edge
		010	Interrupt on falling edge
		011	Interrupt on any edge
		100	Interrupt on level 0
		101	Interrupt on level 1
		110	Reserved
		111	Reserved
10	Wakeup	0	Not a wakeup interrupt
		1	Configure as wakeup interrupt
14:11	GPIO Config	See ref.	GPIO functionality configuration
27:15	Reserved	N/A	Reserved
28	Use for Init	0	Do not use this value as initialization value.

Bit	Field	Value	Description
		1	Use this value as initialization value.
29	Input/Output	0	Configure direction of GPIO as output.
		1	Configure direction of GPIO as input.
30	Output Config	0	Configure GPIO to output low if direction is set to output.
		1	Configure GPIO to output high if direction is set to output.
31	Valid	0	Entry is invalid.
		1	Entry is valid.

APIs exported by QCOM GPIOs are used to enable or disable any peripherals in the system. To enable a peripheral, application invokes the

qcom_gpio_apply_peripheral_configuration(peripheral_id, FALSE) API, which internally gets the GPIO numbers for the given interface from the interface-to-GPIO mapping table. The obtained GPIO numbers are then configured to active mode using the configuration information of the GPIO pins from the active table. Similarly for disabling a peripheral, when an application invokes **qcom_gpio_apply_peripheral_configuration(peripheral_id, TRUE)** API, the API obtains and configures the GPIO numbers using the information from the inactive table.

3.15.2 Pin-based active configuration set

The pin-based active configuration set is an enhancement to the current GPIO implementation, and is backward compatible with previous releases. This mechanism does not require any change to the existing tunable input text file, but requires dynamic addition of peripheral pin configurations using new API(s) if any GPIO pin(s) are to be shared across multiple peripherals.

In pin-based active configuration set, every GPIO pin is perceived as a single node linked-list. The node contains the supported pin configurations (peripheral ID and GPIO active configuration) as tuples, which are dynamically added by the application. The format of GPIO active configuration must be the same as that of the GPIO_ACTIVE_CONFIG parameter of the tunable input text file.

gpio_pin_peripheral_config_t

The tuple of peripheral ID and GPIO active configuration can be passed using the below structure.

```
typedef struct gpio_pin_peripheral_config_s
{
    A_UINT32  peripheral_id;
    A_UINT32  gpio_active_config;
} gpio_pin_peripheral_config_t;
```

gpio_pin_config_head_t

This is an internal structure to the GPIO framework. It is provided only for the understanding of GPIO framework implementation. Applications do not need to directly read or update this data structure. This structure data type is defined for all the GPIO pins and acts as the head of linked list.

```
typedef struct gpio_pin_config_head_s
{
```

```

A_UINT32 supported_peripheral_map; /* upto 32 peripherals */
A_UINT16 active_peripheral_id;
A_UINT16 num_dynamic_configs;
gpio_pin_peripheral_config_t primary_config;
gpio_pin_multi_config_t pLink;
} gpio_pin_config_head_t;

```

- **supported_peripheral_map** – Bitmap of peripheral IDs supported by this pin. When GPIO pin map is requested for a given peripheral ID, the pin map is obtained by noting all the pins having this field.
- **active_peripheral_id** – Current active peripheral ID assigned to a GPIO. When a peripheral is enabled, the GPIOs required for the peripheral are programmed and the active peripheral ID field of all the programmed GPIOs are updated with the enabled peripheral ID.
- **num_dynamic_configs** – Contains the number of dynamic configurations that are added on this GPIO pin.
- **primary_config** – Contains the peripheral ID and GPIO active configuration input read from the tunable GPIO configuration.
- **pLink** – This parameter is of type **gpio_pin_multi_config_t**. The definition of this structure is as follows.

```

typedef struct gpio_pin_multi_config_s
{
    gpio_pin_peripheral_config_t    *pConfigs;
    struct gpio_pin_multi_config_s *pNext;
} gpio_pin_multi_config_t;

```

- **pConfigs** – A pointer to array of **gpio_pin_peripheral_config_t** that is added by the application dynamically. This points to NULL after bootup. At the time of dynamic configuration, memory is allocated for this pointer dynamically. The size of the memory depends on the number of active configurations added by the application.
- **pNext** – This field is not used today and always set to NULL. This field is useful if applications require dynamic GPIO configurations to be added during runtime. Currently it is supported to add only during the initialization time.

3.15.3 Adding pin configurations to the GPIO framework

During the initialization, application needs to add all possible configurations that a pin needs to support. The GPIO configurations can be maintained in the application and passed on to the GPIO framework during initialization. All GPIO pin configurations shall be added in one stretch to achieve memory efficiency.

GPIO framework exports following API to add multiple configurations for a given pin. The prototype of the API is as follows.

```

A_STATUS qcom_gpio_add_alternate_configurations (A_UINT32 pin_num, A_UINT32
    num_configs, gpio_pin_peripheral_config_t *configurations);

```

Using this API, application adds multiple configurations for a given pin. An example is given below showing how to add pin configurations dynamically. In this example, GPIO 7 supports multiple configurations and following are the possible peripheral IDs and the corresponding active mode configurations.

```

gpio_pin_peripheral_config_t gpio_pin_7_configs[] =
{
    { peripheral_id_0, active_config_0 },
    { peripheral_id_1, active_config_1 },
    { peripheral_id_5, active_config_5 },
    { peripheral_id_8, active_config_8 },
}

```

To add this configuration to the framework, the API is called in following manner:

```

qcom_gpio_add_alternate_configurations(7, sizeof
    (gpio_pin_7_configs)/sizeof(gpio_pin_peripheral_config_t),
    gpio_pin_7_configs);

```

The implementation of the **qcom_gpio_add_alternate_configurations()** API inside GPIO framework allocates a memory chunk based on the number of configurations available in the given config and copies the data structure passed on to the API. It also updates all the required parameters in the head of the linked list, namely, `supported_peripheral_map`, `num_dynamic_configs`, and the link pointer.

For a given GPIO pin, application needs to call this API only once with all possible configurations. If this API is called more than once for the same pin, the API will return **A_HW_CONFIG_ERROR** error code. During store-recall, the dynamic active configuration is saved and restored as part of GPIO store recall framework. If application attempts to add dynamic configuration at the time of recall, this API can return **A_EEXIST** error code. Application can choose to ignore the error if the error code returned is **A_EEXIST**. But care should be taken by application since **A_EEXIST** will be returned even if configuration is already available in GPIO framework for just one of the peripherals among set of configurations for multiple peripherals. For example, if dynamic configurations are added for peripherals 'x', 'y' and 'z' and the GPIO framework already contains configuration for peripheral 'x', in the case **A_EEXIST** error code will be returned even though configurations for peripherals 'y' and 'z' are not present in GPIO framework. Applications should make sure that no two configurations are added for any peripheral(s) on any GPIO pin(s).

[Figure 3-4](#) shows how the GPIO data structure looks like after multiple configurations are added for a GPIO pin. The GPIO pins are perceived as an array of single-node linked list.

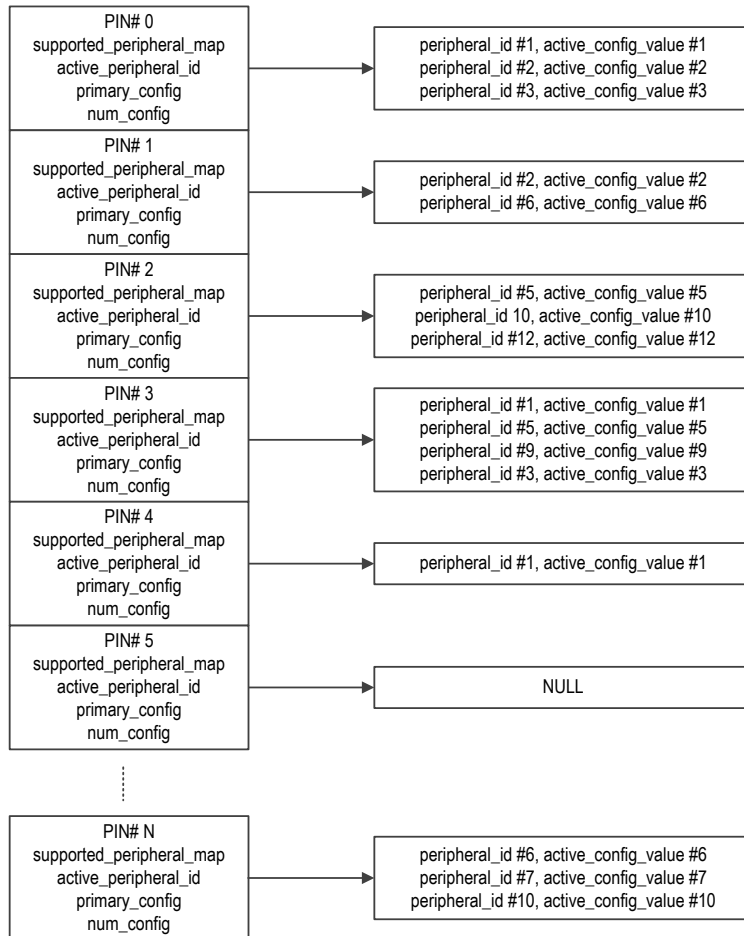


Figure 3-4 GPIO data structure after configuration

3.15.4 GPIO mode of operation

In many use cases, GPIOs are programmed and used as a mere general purpose input/output rather than peripheral access. Every pin operating in GPIO mode is identified by a unique peripheral ID. A constant peripheral ID **QCOM_PERIPHERAL_ID_GPIO** is assigned for GPIO mode by overloading the most significant byte of the peripheral ID with (GPIO pin number + 1). Internal to the GPIO framework, the peripheral ID for GPIO mode is just a constant, while for application it is the combination of constant GPIO peripheral ID and the GPIO pin number.

For example, assume that peripheral ID for GPIO mode of operation is 11 (0xB) which is constant irrespective of the pin number. For applications, the peripheral ID for GPIO pin 0 to operate in GPIO mode can be 0x0100000B. This applies to other pins as follows:

0x0200000B – GPIO mode for pin 1

0x0300000B – GPIO mode for pin 2

0x0400000B – GPIO mode for pin 3

To get the peripheral ID for a hardware pin to be configured to GPIO mode, applications can use the **QCOM_PERIPHERAL_ID_GPIO***n*(*x*) macro which takes hardware pin number as the input and returns the peripheral ID as the output. The 'x' refers to the hardware pin number for which the GPIO peripheral ID is to be obtained.

In the GPIO mode of operation, the pin must be allowed to change its active configuration parameters such as direction and interrupt edge. Therefore multiple active configuration values are possible when a pin operates in GPIO mode. To support this feature, the GPIO framework allows updating the active configuration parameters when a pin is operating in GPIO mode.

3.15.5 Interrupt registration

To allow applications or application library to register interrupts to GPIO pins, GPIO pins are exported in a logical level rather than as a hardware pin because every pin can be multiplexed to achieve various functionalities. Therefore, logical functional pins are exported to the application and internally derive the hardware pin based on the peripheral configuration programmed at that time.

To register an interrupt handler with any logical functional pin, application (or application library) needs to follow the steps below.

1. A header file **qcom_gpio_interrupts.h** is exported which contains all the logical functional pin definitions as macros. These are mere numbers and does not have any correlation to the hardware pins that supports this feature.
2. GPIO frameworks internally maintains a table which contains possible configuration mappings for all the hardware pins. This table serves as a lookup table for GPIO framework to obtain the hardware pin for the given logical pin.
3. Application should obtain the hardware pin number for the logical pin using the API below. The obtained hardware pin should be used to register an interrupt handler, see step 4.

```
A_STATUS qcom_gpio_get_interrupt_pin_num(A_UINT32 interrupt_num,
A_UINT32 *pPinNum);
```

- **interrupt_num** – GPIO functional pin for which the hardware pin number is to be obtained
 - **pPinNum** – Hardware pin number filled by the GPIO framework for the requested interrupt number. Value returned in this parameter must be used only if the API returns A_OK.
4. The obtained hardware pin must be used with interrupt registration APIs in order to register an interrupt handler for any of the logical pin. Interrupt registration on a hardware pin is allowed only if the pin is configured to the same peripheral ID as that of the logical pin's peripheral ID. In other words, the peripheral corresponding to the logical pin should be enabled before registering an interrupt handler.
 5. Before disabling a peripheral, application must make sure that all the interrupt handlers associated with that peripheral are de-registered.

3.15.6 Suspend/resume

As the GPIO configurations are dynamically added and updated (in case of GPIO mode), it is necessary to save and restore the GPIO configurations across suspend and resume of QCA4010/QCA4012. To achieve this, GPIO framework internally registers itself with the wakeup manager and performs save and restore of necessary data during suspend and resume operations.

The data to be saved and restored includes:

- GPIO active configuration table

- Registered interrupt handlers
- Reprogramming values of GPIO registers that are saved before suspend
- Dynamic GPIO data that are added as part of initialization

If the application attempts to add the dynamic configuration at system resume, `qcom_gpio_add_alternate_configurations()` can return `A_EEXIST` error code in the failure. Application can choose to ignore if the returned error code is `A_EEXIST` as the GPIO framework has already added the dynamic configuration as part of GPIO resume.

4 P2P Demo

This chapter describes the setup and test procedures for the P2P demo application. The required equipment includes:

- Development kit
- P2P demo application preloaded to Wi-Fi module flash

The P2P demo application implements a P2P command set for access from the serial console. The P2P command set provides an interface to configure and control the QCA4010/QCA4012 Wi-Fi chipset for connecting with a P2P device.

- Ubuntu 9.10 laptop as a P2P peer device

The P2P peer device is a standard Wi-Fi P2P certification testbed.

- Qualcomm Technologies XB92 PCI Express Card
- Qualcomm Technologies P2P Linux reference client

4.1 P2P commands

This section describes the commands that can be used to operate the Wi-Fi module in P2P mode (Wi-Fi Direct mode of operation).

P2P commands have the syntax as below. The option <p2p_specific_commands> is summarized in [Table 4-1](#) along with a brief description.

```
wmiconfig --p2p <p2p_specific_commands>
```

Table 4-1 P2P command options

Command	Description
on/off	Enable/disable P2P mode of operation.
status	Print the status of P2P device.
cancel	Cancel all the ongoing P2P operations.
nodelist	Display the results of P2P find operation.
find	Initiate P2P find operation.
list_network	Display the list of persistent P2P connections that are saved in the persistent media.
autogo	Start P2P device in Autonomous Group Owner mode.
auth	Authenticate/Reject a connection request from a given peer MAC address using the given WPS configuration method.
connect	Initiate connection request with a given peer MAC address using given WPS configuration method.
invite	Invite the remembered persistent peer to for the group.

Command	Description
invite_auth	Authenticate invitation request.
join	Join a P2P client to an existing P2P Group Owner with the given interface MAC address and WPS config method.
prov	Provision the WPS configuration method between the DUT and the peer.
set	Generic command to set multiple P2P parameters.
setconfig	Configure specific parameters for a P2P device.
start_pers	SSID of GO in persistent group
passphrase	Set the passphrase.
setoppps	Enable and set the parameters of OppPS for GO.
setnoa	Enable and set the parameters of NoA for GO.
get	Get the passphrase that the current P2P connection is using.
listen	Initiate P2P listen process.

A brief description of all the available P2P commands and their associated parameters are described in the following sections.

4.1.1 on/off

Syntax	<i>wmiconfig --p2p <on/off></i>		
Description	Enable/disable P2P mode of operation.		
Parameters	<i>on/off</i>	on	Enable P2P mode of operation.
		off	Disable P2P mode of operation.

4.1.2 status

Syntax	<i>wmiconfig --p2p status</i>
Description	Print the status of P2P device.
Parameters	None

4.1.3 cancel

Syntax	<i>wmiconfig --p2p cancel</i>
Description	Cancel all ongoing P2P operation. P2P is put to the same state after it is turned on.
Parameters	None

4.1.4 nodelist

Syntax	<i>wmiconfig --p2p nodelist</i>
Description	Display the results of P2P find operation. This command is used after <i>find</i> .
Parameters	None

4.1.5 find

Syntax	<i>wmiconfig --p2p find <channel_options> <timeout></i>		
Description	Initiate P2P find operation. The <i>find</i> operation includes a sequence of SEARCH/LISTEN phase across the channels to look for available P2P peers.		
Parameters	<i>channel_options</i>	full	Scan all the channels from regulatory domain channel list.
		social	Scan only the social channels (Channel 1, 6 and 11 in 2.4GHz band). This is the default option for the <i>find</i> command.
		prog	Continue channel scan from the last scanned channel index. Channel list is obtained from regulatory domain list.
	timeout (in seconds)	When the timeout period expires, the find operation is stopped. Default value is 60 seconds.	

4.1.6 list_network

Syntax	<i>wmiconfig --p2p list_network</i>
Description	Display the list of persistent P2P connections that are saved in the persistent media.
Parameters	None

4.1.7 autogo

Syntax	<i>wmiconfig --p2p autogo [persistent]</i>		
Description	Start P2P device in Autonomous Group Owner mode. This command can be used to start P2P device as a group owner without the need for the Group Owner negotiation procedure. In Auto-GO mode, the default SSID is DIRECT-GO and the default passphrase is 1234567890. The default settings cannot be changed.		
Parameters	<i>persistent</i>	(Optional) This parameter indicates whether the start of autogo is a persistent connection. Persistent connections are remembered in a persistent media and can be re-invoked using P2P invitation request.	

4.1.8 auth

Syntax	<i>wmiconfig --p2p auth <peer_mac> <wps_method> [pin-num] [persistent]</i>		
Description	Authenticate/Reject a connection request from a given peer MAC address using the given WPS configuration method.		
Parameters	<i>peer_p2p_dev_mac</i>	Device MAC address of the P2P peer to accept or reject	
	<i>wps_method</i>	push	Use WPS PBC method for authentication.
		display	Use WPS DISPLAY method for authentication.
		keypad	Use WPS KEYPAD method for authentication.
		deauth	Reject the authentication request for the given peer_mac address.
	<i>pin-num</i>	(Optional) This parameter is set only if wps_method is keypad. The pin number is used for WPS keypad authentication.	

	<i>persistent</i>	(Optional) This parameter indicates whether the connection with given peer_mac address is a persistent connection. Persistent connections are remembered in a persistent media and can be reinvoked using P2P invitation request.
--	-------------------	---

4.1.9 connect

Syntax	<i>wmiconfig --p2p connect <peer_p2p_dev_mac> <wps_method> [pin-num] [persistent]</i>		
Description	Initiate connection request with a given peer MAC address using given WPS configuration method.		
Parameters	<i>peer_p2p_dev_mac</i>	Device MAC address of the P2P peer to connect with	
	<i>wps_method</i>	push	Use WPS PBC method for authentication.
		display	Use WPS DISPLAY method for authentication.
		keypad	Use WPS KEYPAD method for authentication.
	<i>pin-num</i>	(Optional) This parameter is set only if wps_method is keypad. The pin number is used for WPS keypad authentication.	
	<i>persistent</i>	(Optional) This parameter indicates whether the connection with given peer_mac address is a persistent connection. Persistent connections are remembered in a persistent media and can be re-invoked using P2P invitation request.	

4.1.10 invite

Syntax	<i>wmiconfig --p2p invite <ssid> <p2p_dev_mac> <wps_method> [persistent]</i>		
Description	Invite a connection from persistent database. Re-invoking a persistent connection enables quicker association because the passphrase is available at database, saving the 8-way WPS handshake.		
Parameters	<i>ssid</i>	SSID of the persistent connection to be re-invoked	
	<i>p2p_dev_mac</i>	P2P device address of the peer that should be re-invoked	
	<i>wps_method</i>	push	Use WPS PBC method for authentication.
		display	Use WPS DISPLAY method for authentication.
		keypad	Use WPS KEYPAD method for authentication.
	<i>persistent</i>	(Optional) This parameter indicates whether the connection with given peer_mac address is a persistent connection. Persistent connections are remembered in a persistent media and can be re-invoked using P2P invitation request.	

4.1.11 invite_auth

Syntax	<i>wmiconfig --p2p invite_auth <peer_p2p_device_addr> [reject]</i>	
Description	Authenticate invitation request.	
Parameters	<i>peer_p2p_device_addr</i>	Device MAC address of the P2P peer for re-invoking GO.
	<i>reject</i>	(Optional) Reject the invitation request.

4.1.12 join

Syntax	<i>wmiconfig --p2p join <peer_p2p_interface_mac> <wps_method> [pin-num] [persistent]</i>		
Description	Join a P2P client to an existing P2P Group Owner.		
Parameters	<i>peer_p2p_interface_mac</i>	GO interface MAC address of the P2P peer to connected with	
	<i>wps_method</i>	push	Use WPS PBC method for authentication.
		display	Use WPS DISPLAY method for authentication.
		keypad	Use WPS KEYPAD method for authentication.
	<i>pin-num</i>	(Optional) This parameter is set only if wps_method is keypad. The pin number is used for WPS keypad authentication.	
	<i>persistent</i>	(Optional) This parameter indicates whether the connection with given peer_mac address is a persistent connection. Persistent connections are remembered in a persistent media and can be re-invoked using P2P invitation request.	

4.1.13 prov

Syntax	<i>wmiconfig --p2p prov <peer_p2p_device_mac> <wps_method></i>		
Description	Provision the WPS configuration method between the DUT and the peer.		
Parameters	<i>peer_p2p_device_mac</i>	Device MAC address of the P2P peer to be provisioned	
	<i>wps_method</i>	push	Use WPS PBC method for authentication.
		display	Use WPS DISPLAY method for authentication.
		keypad	Use WPS KEYPAD method for authentication.

4.1.14 set

Syntax	<i>wmiconfig --p2p set [gointent <intent_val>] [p2pmode <mode>] [postfix <postfix_string>] [inrabss <inrabss_val>] [name <name_string>]</i>		
Description	Generic command to set multiple P2P parameters		
Parameters	<i>intent_val</i>	Integer value to be set as GO intent. This GO intent value is used in P2P GO Negotiation Request/Response frames. The value ranges from 0 to 15.	
	<i>mode</i>	p2pdev	Set P2P operating mode as P2P Device.
		p2pclient	Set P2P operating mode as P2P Client.
		p2pgo	Set P2P operating mode as P2P Group Owner.
	<i>postfix_string</i>	The string to be added to the SSID suffix when operating as P2P GO. P2P GO SSID starts with the string 'DIRECT-'. The string entered in this parameter is appended to 'DIRECT-'. Max length of this string is 22.	
	<i>inrabss_val</i>	0	Disable intra-BSS transition distribution in capability field of P2P IE.
		1	Enable intra-BSS transition distribution in capability field of P2P IE.
	<i>name</i>	P2P device name to be set for the device	

4.1.15 setconfig

Syntax	<i>wmiconfig --p2p setconfig <go_intent> <listen_ch> <operating_ch> <country> <node_timeout></i>	
Description	Set the parameters that are not covered by the <i>set</i> command.	
Parameters	<i>go_intent</i>	Integer value to be set as GO intent. This GO intent value is used in P2P GO Negotiation Request/Response frames. The value ranges from 0 to 15.
	<i>listen_ch</i>	The channel to listen during P2P <i>find</i> operation. The DUT remains on this channel during LISTEN phase of the <i>find</i> operation for random duration. During this phase, DUT responds to probe requests from peer P2P devices.
	<i>operating_ch</i>	The preferred channel to operate the P2P Group
	<i>country</i>	No longer used. Country code needs to be set by board data file or tunables.
	<i>node_timeout</i>	Timeout value (in seconds) for each node. After the timeout period expires, the nodes are deleted from P2P find results.

4.1.16 start_pers

Syntax	<i>wmiconfig --p2p start_pers <SSID></i>	
Description	Re-invoke a persistent group where the DUT is GO.	
Parameters	<i>SSID</i>	SSID of GO in persistent group.

4.1.17 passphrase

Syntax	<i>wmiconfig --p2p passphrase <passphrase> <SSID></i>	
Description	Set the passphrase	
Parameters	<i>passphrase</i>	The passphrase to use.
	<i>SSID</i>	SSID to use.

4.1.18 setoppps

Syntax	<i>wmiconfig --p2p setoppps <enable> <ctwin></i>	
Description	Enable and Set the parameters of OppPS for GO.	
Parameters	<i>enable</i>	Enable or disable OppPS feature for GO.
	<i>ctwin</i>	CTWindow parameter in OppPS for GO, in units of TU (1024 microseconds).

4.1.19 setnoa

Syntax	<i>wmiconfig --p2p setnoa <count> <start> <duration> <interval></i>	
Description	Enable and Set the parameters of NoA for GO.	
Parameters	<i>count</i>	1-255. Indicate the number of absence interval for GO.

	<i>start</i>	In units of millisecond. Indicate the offset time after the next TBTT, to calculate the absolute start time of GO's first absence period.
	<i>duration</i>	Indicate the maximum duration in units of milliseconds that the P2P GO can remain absent in one NoA interval.
	<i>interval</i>	Indicate the length of the Notice of Absence interval in units of milliseconds.

4.1.20 get

Syntax	<code>wmiconfig --p2p get passphrase</code>
Description	Get the passphrase that the current P2P connection is using.
Parameters	None

4.1.21 listen

Syntax	<code>wmiconfig --p2p listen <timeout></code>
Description	Initiate P2P listen process.
Parameters	<i>timeout</i> In units of second. When the timeout period expires, the listen operation is stopped. Default value is 30 seconds.

4.2 P2P demo setup

The setup includes a development kit connected to a PC via usbWiggler ONCE (JP12), power port (J3) and console port (J1). A console application running on the PC1 is used to input command for P2P configuration and connection. Using these commands, a Wi-Fi link can be created between the P2P peer device (PC2) and development kit in different scenarios. [Figure 4-1](#) shows an example configuration.

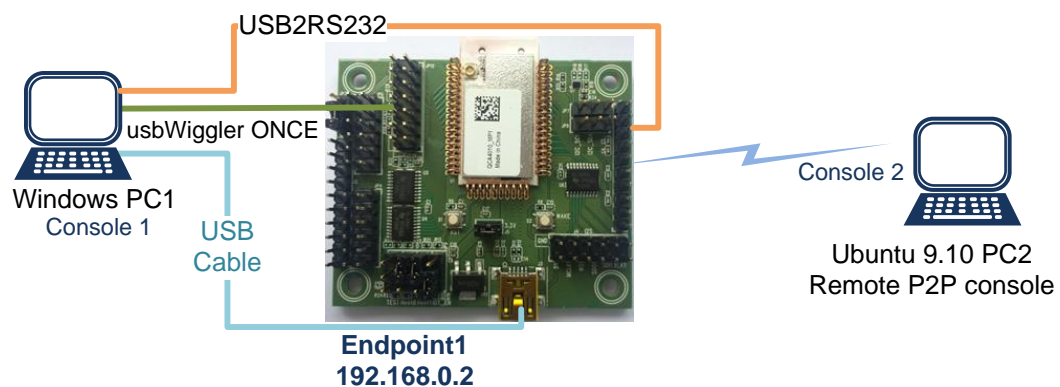


Figure 4-1 Sample system configuration for P2P demonstration (RB01+RB02)

4.2.1 Windows PC1 setup

1. Plug the usbWiggler ONCE adapter to PC1.
2. Connect the usbWiggler ONCE adapter cable to JP12 on the RB01/RB02 development platform.
3. Use a USB2RS232 adapter cable to connect PC1 and the RB01/RB02 through J1.
4. Use a USB cable to connect PC1 and the RB01/RB02 through J3. Wait for several seconds to have the RB01/RB02 start up.
5. Start a serial terminal application from PC1 and select the lower COM port to connect using the port setting: 115200, 8, n, 1, no flow control.

4.2.2 Ubuntu PC2 setup

1. Plug the XB92 PCI Express card to PC2 and install the Qualcomm Technologies P2P Linux client.

2. Edit the configuration file /home/atheros/Atheros-P2P/wpa-default.conf.

```
Ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=admin
device_name=Atheros Wi-Fi Direct
device_type=1-0050F204-1
p2p_go_intent=9
p2p_listen_reg_class=81
p2p_listen_channel=6
p2p_oper_reg_class=115
p2p_oper_channel=6
persistent_reconnect=1
driver_param=shared_interface=wlan0
```

3. Run the following commands to start a P2P interface on the XB92 card.

```
sudo -i
cd /home/atheros/Atheros-P2P/
rmmod ath9k
rmmod ath
sh start-wpas-newma.sh
./wpa_cli
```

The following startup log is printed.

```
atheros@Atheros-P2P:~$ sudo -i
root@Atheros-P2P:~# cd /home/atheros/Atheros-P2P/
root@Atheros-P2P:/home/atheros/Atheros-P2P# rmmod ath9k
root@Atheros-P2P:/home/atheros/Atheros-P2P# rmmod ath
root@Atheros-P2P:/home/atheros/Atheros-P2P# sh start-wpas-newma.sh
stop: Unknown instance:
wpa_supplicant: no process found
stop: Unknown instance:
dnsmasq: no process found
```

```

ERROR: Module umac does not exist in /proc/modules
ERROR: Module ath_dev does not exist in /proc/modules
ERROR: Module ath_rate_atheros does not exist in /proc/modules
ERROR: Module ath_hal does not exist in /proc/modules
ERROR: Module asf does not exist in /proc/modules
ERROR: Module adf does not exist in /proc/modules
wlan0
wlan1
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
wpa_cli v0.7.3-athr-p2p-release-12
Selected interface 'wlan1'
Interactive mode
>

```

4. View the control interface.

The control interface of the P2P peer device is wlan0 which is the initial interface of XB92. The MAC address of the control interface is 00:03:7f:xx:xx:xx,.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ifconfig wlan0
wlan0      Link encap:Ethernet  HWaddr 00:03:7f:10:51:38
            inet6 addr: fe80::203:7fff:fe10:5138/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

5. View the device interface.

The device interface of the P2P peer device is wlan1 which is generated from wlan0. The MAC address of the device interface is 02:03:7f:xx:xx:xx.

Device interface is available only after the P2P peer device brings up successfully. Device-related operation sets such as wpa_cli, Device Discovery, and Group Formation are executed on this interface.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ifconfig wlan1
wlan1      Link encap:Ethernet  HWaddr 02:03:7f:10:51:38
            inet6 addr: fe80::3:7fff:fe10:5138/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:4 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

6. View the group interface.

The group interface of P2P peer device is wlan2 which is generated from wlan1. This interface is used for group operations. The MAC address of the group interface is 12:03:7f:xx:xx:xx .

Group interface is available only after P2P peer device starts a P2P Group. Group-related operation sets such as Invitation and Group Operations, as well as the traffics are executed on this interface.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli p2p_group_add
Selected interface 'wlan1'
OK
root@Atheros-P2P:/home/atheros/Atheros-P2P# ifconfig wlan2
wlan2      Link encap:Ethernet  HWaddr 12:03:7f:10:51:38
            inet addr:192.168.1.10  Bcast:192.168.1.255
Mask:255.255.255.0
            inet6 addr: fe80::1003:7fff:fe10:5138/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:2 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

7. Configure IP address for the group interface.

Two approaches are available to configure the group interface IP address.

□ Linux internal command

```
ifconfig wlan2 192.168.1.xx netmask 255.255.255.0
```

Printed log:

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ifconfig wlan2
192.168.1.102 netmask 255.255.255.0
root@Atheros-P2P:/home/atheros/Atheros-P2P# ifconfig wlan2
wlan2      Link encap:Ethernet  HWaddr 12:03:7f:10:51:38
            inet addr:192.168.1.102  Bcast:192.168.1.255
Mask:255.255.255.0
            inet6 addr: fe80::1003:7fff:fe10:5138/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:3 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

□ Configure /home/atheros/Atheros-P2P/p2p-action.sh before starting P2P Group.

```

if [ "$CMD" = "P2P-GROUP-STARTED" ]; then
    GIFNAME=$3
    if [ "$4" = "GO" ]; then
        kill_daemon dhclient /var/run/dhclient-$GIFNAME.pid
        rm /var/run/dhclient.leases-$GIFNAME
        kill_daemon dnsmasq /var/run/dnsmasq.pid-$GIFNAME
        ifconfig $GIFNAME 192.168.1.10 up
        dnsmasq -x /var/run/dnsmasq.pid-$GIFNAME \
            -i $GIFNAME \
            -F192.168.1.100,192.168.1.200
    fi
    if [ "$4" = "client" ]; then
        kill_daemon dhclient /var/run/dhclient-$GIFNAME.pid
        rm /var/run/dhclient.leases-$GIFNAME
        kill_daemon dnsmasq /var/run/dnsmasq.pid-$GIFNAME
    fi
fi

```

```

dhclient -pf /var/run/dhclient-$GIFNAME.pid \
        -cf /home/atheros/Atheros-P2P/dhclient-p2p.conf \
        -lf /var/run/dhclient.leases-$GIFNAME \
        -nw \
        $GIFNAME
fi
fi

```

4.3 P2P find and provision

4.3.1 Wi-Fi module using PUSH method

4.3.1.1 Provisioning Discovery from the Wi-Fi module

1. Run the following commands on the P2P peer device.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find

```

Following are the logs.

```

# LOGs of p2p_find
[LOG]OK
[LOG]> <2>P2P-DEVICE-FOUND 84:5d:d7:40:af:01
p2p_dev_addr=84:5d:d7:40:af:01 pri_dev_type=6-0050F204-0 name='IOE-DEV'
config_methods=0x188 dev_capab=0x22 group_capab=0x0
# LOGs of receiving provision form the Wi-Fi module
[LOG]<2>P2P-PROV-DISC-PBC-REQ 84:5d:d7:40:af:01
p2p_dev_addr=84:5d:d7:40:af:01 pri_dev_type=6-0050F204-0 name='IOE-DEV'
config_methods=0x188 dev_capab=0x22 group_capab=0x0

```

2. Run the following commands on the Wi-Fi module.

```

wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
shell> wmiconfig --p2p nodelist

```

Following are the logs.

```

# LOGs of p2p nodelist
[LOG]p2p_dev_name      : Atheros Wi-Fi Direct
[LOG]wps_method        : 0x00000000
[LOG]config_methods    : 0x0188
[LOG]persistent_grp    : 0x00
[LOG]p2p_int_addr      : 0:0:0:0:0:0
[LOG]p2p_dev_addr      : 2:3:7f:10:51:38
[LOG]p2p_dev_capab     : 0x23
[LOG]p2p_grp_capab     : 0x00

```

3. Run the following command.

```

shell> wmiconfig --p2p prov 02:03:7f:10:51:38 push

```

Following are the logs.

```
# LOGs of sending provision to the P2P peer device
[LOG]peer mac : 2:3:7f:10:51:38
[LOG]wps method : 0x0080
[LOG]provisional disc resp - Push Button
```

4.3.1.2 Provisioning Discovery from the peer device

1. Run the following commands on the Wi-Fi module.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
```

Following are the logs.

```
# LOGs of receiving provision from the peer device
[LOG]SA : 2:3:7f:10:51:38
[LOG]wps method : 0x0080
[LOG]provisional disc request - Push Button
```

2. Run p2p_find on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```

Following are the logs.

```
# LOGs of p2p_find
[LOG]OK
[LOG]> <2>P2P-DEVICE-FOUND 84:5d:d7:40:af:01
p2p_dev_addr=84:5d:d7:40:af:01 pri_dev_type=6-0050F204-0 name='IOE-DEV'
config_methods=0x188 dev_capab=0x22 group_capab=0x0
```

3. Run p2p_peers on the peer device.

```
p2p_peers
```

Following are the logs.

```
# LOGs of p2p_peers
[LOG]84:5d:d7:40:af:01
```

4. Run p2p_prov_disc on the peer device.

```
p2p_prov_disc 84:5d:d7:40:af:01 pbc
```

Following are the logs.

```
# LOGs of sending provision to the Wi-Fi module
[LOG]OK
[LOG]<2>P2P-PROV-DISC-PBC-RESP 84:5d:d7:40:af:01
```

4.3.2 Development kit using KEYPAD or DISPLAY method

4.3.2.1 Provisioning Discovery on both sides

1. Run the following commands on the Wi-Fi module.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
```

2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
```
3. Send provisioning request from the development kit to see whether KEYPAD is supported on the peer device.

```
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 keypad
```
4. Send provisioning request from the peer device to see whether KEYPAD and DISPLAY is supported on the development kit.

```
p2p_prov_disc 84:5d:d7:40:af:01 keypad
p2p_prov_disc 84:5d:d7:40:af:01 display
```
5. Send provisioning request from the development kit to see whether DISPLAY is supported on the peer device.

```
wmiconfig --p2p prov 02:03:7f:10:51:38 display
```

4.4 P2P connection with PUSH on the development kit

4.4.1 Development kit as P2P Client or P2P GO

4.4.1.1 Development kit connecting to peer device

1. Run the following commands on the development kit.

```
wmiconfig --p2p on
```

 - For P2P Client, run:

```
wmiconfig --p2p setconfig 0 6 6 US 300
```
 - For P2P GO, run:

```
wmiconfig --p2p setconfig 15 6 6 US 300
```

```
wmiconfig --p2p find
```
2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
```
3. Run the following commands on the development kit.

```
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 push
```
4. Run the following commands on the peer device.

```
p2p_connect 84:5d:d7:40:af:01 pbc auth
```
5. Run the following commands on the development kit.

```
wmiconfig --p2p connect 02:03:7f:10:51:38 push
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
```

Logs on the development kit

```
# LOGs of send "p2p connect" from the development kit to the peer device
[LOG]dev=0, go intend: 0
# LOGs of Negotiation on the development kit
[LOG]shell> P2P GO neg result :
[LOG]Status      :SUCCESS
[LOG]P2P Role    :Client
[LOG]SSID        :DIRECT-TA
[LOG]Channel     :2437
[LOG]WPS Method  :PBC
[LOG]WPS connecting ...
[LOG]Erri:sndfrm 44c88c
[LOG]Erri:sndfrm 44c86c
[LOG]Erri:sndfrm 44c8ac
[LOG]Erri:sndfrm 44c8cc
[LOG]Disconnected from DIRECT-TA
[LOG]noise_floor = -115
[LOG]Connected to DIRECT-TA,
[LOG]Erri:sndfrm 44c8cc
[LOG]Erri:sndfrm 44c8ac
# Logs of PING replyment from the peer device
[LOG]Ping reply from 192.168.1.10: time<1ms
```

Logs on the peer device

```
# Logs of p2p_connect auth on the peer device
[LOG]OK
# LOGs of Negotiation on the peer device
[LOG]<2>P2P-GO-NEG-SUCCESS
[LOG]<2>P2P-GROUP-FORMATION-SUCCESS
[LOG]<2>P2P-GROUP-STARTED wlan2 GO ssid="DIRECT-TA" freq=2437
passphrase="r5ETzjrw" go_dev_addr=02:03:7f:10:51:38
```

4.4.1.2 Peer device connecting to the development kit

1. Run the following commands on the development kit.

```
wmiconfig --p2p on
```

- For P2P Client, run:

```
wmiconfig --p2p setconfig 0 6 6 US 300
```

- For P2P GO, run:

```
wmiconfig --p2p setconfig 15 6 6 US 300
```

```
wmiconfig --p2p find
```

2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
```

3. Run the following commands on the development kit.

- ```
wmiconfig --p2p nodelist
```
4. Run the following commands on the peer device.
 

```
p2p_prov_disc 84:5d:d7:40:af:01 pbc
```
  5. Run the following commands on the development kit.
 

```
wmiconfig --p2p auth 02:03:7f:10:51:38 push
```
  6. Run the following commands on the peer device.
 

```
p2p_connect 84:5d:d7:40:af:01 pbc
```
  7. Run the following commands on the development kit.
 

```
wmiconfig --ipdhcp
wmiconfig --ipconfig
ping 192.168.1.10
```

### Logs on the development kit

```
LOGs of set "p2p auth" on the development kit
[LOG]dev=0, go intend: 0
LOGs of Negotiation on the development kit
[LOG]shell> P2P GO neg result :
[LOG]Status :SUCCESS
[LOG]P2P Role :Client
[LOG]SSID :DIRECT-Ng
[LOG]Channel :2437
[LOG]WPS Method :PBC
[LOG]WPS connecting ...
[LOG]Erri:sndfrm 44c84c
[LOG]Erri:sndfrm 44c82c
[LOG]Disconnected from DIRECT-Ng
[LOG]noise_floor = -114
[LOG]Connected to DIRECT-Ng,
[LOG]Erri:sndfrm 44c82c
[LOG]Erri:sndfrm 44c84c
Logs of getting DHCP from the peer device
[LOG]shell> _dhc_setip Got IP address c0a801b2
Logs of check IP on RB01/RB02
[LOG] mac addr = 84:5d:d7:40:af:1
[LOG]IP:192.168.1.178
[LOG] Mask:255.255.255.0
[LOG], Gateway:192.168.1.10
[LOG], Dns:10.1.168.192
[LOG]Link-local IPv6 Address : FE80::865D:D7FF:FE40:AF01/64
[LOG]Global IPv6 Address : ::
[LOG]Default Gateway : ::
[LOG]Global IPv6 Address 2 : ::
Logs of PING
[LOG]Ping reply from 192.168.1.10: time<1ms
```

## Logs on the peer device

```
Logs of p2p_connect on the peer device
[LOG]OK
LOGs of Negotiation on the peer device
[LOG]<2>P2P-GO-NEG-SUCCESS
[LOG]<2>P2P-GROUP-FORMATION-SUCCESS
[LOG]<2>P2P-GROUP-STARTED wlan2 GO ssid="DIRECT-Ng" freq=2437
passphrase="4gR703Af" go_dev_addr=02:03:7f:10:51:38
```

## 4.5 P2P connection with KEYPAD on the development kit

### 4.5.1 Development kit as P2P Client or P2P GO

#### 4.5.1.1 Development kit connecting to peer device

1. Run the following commands on the development kit.
 

```
wmiconfig --p2p on
```

  - For P2P Client, run:
 

```
wmiconfig --p2p setconfig 0 6 6 US 300
```
  - For P2P GO, run:
 

```
wmiconfig --p2p setconfig 15 6 6 US 300
```

```
wmiconfig --p2p find
```
2. Run the following commands on the peer device.
 

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
```
3. Run the following commands on the development kit.
 

```
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 display
```
4. Run the following commands on the peer device.
 

```
p2p_connect 84:5d:d7:40:af:01 12345670 display auth
```
5. Run the following commands on the development kit.
 

```
wmiconfig --p2p connect 02:03:7f:10:51:38 keypad 12345670
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
```

#### 4.5.1.2 Peer device connecting to the development kit

1. Run the following commands on the development kit.
 

```
wmiconfig --p2p on
```

  - For P2P Client, run:
 

```
wmiconfig --p2p setconfig 0 6 6 US 300
```
  - For P2P GO, run:

- ```

    wmicongfig --p2p setconfig 15 6 6 US 300
    wmicongfig --p2p find

```
2. Run the following commands on the peer device.


```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers

```
 3. Run the following commands on the development kit.


```

wmicongfig --p2p nodelist

```
 4. Run the following commands on the peer device.


```

p2p_prov_disc 84:5d:d7:40:af:01 keypad

```
 5. Run the following commands on the development kit.


```

wmicongfig --p2p auth 02:03:7f:10:51:38 keypad 12345670

```
 6. Run the following commands on the peer device.


```

p2p_connect 84:5d:d7:40:af:01 12345670 display

```
 7. Run the following commands on the development kit.


```

wmicongfig --ipdhcp
wmicongfig --ipconfig
ping 192.168.1.10

```

4.6 P2P connection with DISPLAY on the development kit

4.6.1 Development kit as P2P Client or P2P GO

4.6.1.1 Development kit connecting to peer device

1. Run the following commands on the development kit.


```

wmicongfig --p2p on

```

 - For P2P Client, run:


```

wmicongfig --p2p setconfig 0 6 6 US 300

```
 - For P2P GO, run:


```

wmicongfig --p2p setconfig 15 6 6 US 300
wmicongfig --p2p find

```
2. Run the following commands on the peer device.


```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers

```
3. Run the following commands on the development kit.


```

wmicongfig --p2p nodelist

```
4. Run the following commands on the peer device.


```

p2p_prov_disc 84:5d:d7:40:af:01 pbc

```
5. Run the following commands on the development kit.

```
wmiconfig --p2p prov 02:03:7f:10:51:38 keypad
```

6. Run the following commands on the peer device.

```
p2p_connect 84:5d:d7:40:af:01 12345670 keypad auth
```

7. Run the following commands on the development kit.

```
wmiconfig --p2p connect 02:03:7f:10:51:38 display 12345670
```

```
wmiconfig --ipdhcp
```

```
wmiconfig --ipconfig
```

```
ping 192.168.1.10
```

4.6.1.2 Peer device connecting to the development kit

1. Run the following commands on the development kit.

```
wmiconfig --p2p on
```

- For P2P Client, run:

```
wmiconfig --p2p setconfig 0 6 6 US 300
```

- For P2P GO, run:

```
wmiconfig --p2p setconfig 15 6 6 US 300
```

```
wmiconfig --p2p find
```

2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
```

```
p2p_find
```

```
p2p_peers
```

3. Run the following commands on the development kit.

```
wmiconfig --p2p nodelist
```

4. Run the following commands on the peer device.

```
p2p_prov_disc 84:5d:d7:40:af:01 display
```

5. Run the following commands on the development kit.

```
wmiconfig --p2p auth 02:03:7f:10:51:38 display 12345670
```

6. Run the following commands on the peer device.

```
p2p_connect 84:5d:d7:40:af:01 12345670 keypad
```

7. Run the following commands on the development kit.

```
wmiconfig --ipdhcp
```

```
wmiconfig --ipconfig
```

```
ping 192.168.1.10
```

4.7 Autonomous GO on the development kit

4.7.1 Peer device joining Auto GO on the development kit

4.7.1.1 Peer device with PUSH method

1. Run the following commands on the development kit.

```

wmiconfig --p2p on
wmiconfig --p2p setconfig 15 6 6 US 300
wmiconfig --p2p autogo

```

2. Run the following commands on the peer device.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
p2p_prov_disc 84:5d:d7:40:af:01 pbc
p2p_connect 84:5d:d7:40:af:01 pbc join

```

3. Run the following commands on the development kit.

```

wmiconfig --wps 1 push
ping 192.168.1.100

```

Logs on the development kit

```

# Logs of starting an AutoGO on the development kit
[LOG]Connected to DIRECT-cO,
[LOG]Erri:sndfrm 44c80c
[LOG]Erri:sndfrm 44c82c
[LOG]Erri:sndfrm 44c84c
# Logs of accept of Join from the peer device (the same with ACK of Prov)
[LOG]SA : 2:3:7f:10:51:38
[LOG]wps method : 0x0080
[LOG]provisional disc request - Push Button
# Logs of formation
[LOG]Error in starting the task ...
[LOG]Disconnected from DIRECT-cO
[LOG]noise_floor = -112
[LOG]Connected to DIRECT-cO,
[LOG]Error in starting the task ...
[LOG]Error in starting the task ...
[LOG]Error in starting the task ...
# Logs of release DHCP to the peer device automatically
[LOG]shell> ip status 6401a8c0 1

```

Logs on the peer device

```

# Logs of formation
[LOG]<2>P2P-PROV-DISC-PBC-RESP 84:5d:d7:40:af:01
[LOG]<2>P2P-GROUP-FORMATION-SUCCESS
[LOG]<2>P2P-GROUP-STARTED wlan2 client ssid="DIRECT-cO" freq=0
passphrase="1234567890" go_dev_addr=84:5d:d7:40:af:01

```

4.7.1.2 Peer device with KEYPAD method

1. Run the following commands on the development kit.

```

wmiconfig --p2p on
wmiconfig --p2p setconfig 15 6 6 US 300
wmiconfig --p2p autogo

```

2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
p2p_prov_disc 84:5d:d7:40:af:01 display
p2p_connect 84:5d:d7:40:af:01 12345670 keypad join
```

3. Run the following commands on the development kit.

```
wmiconfig --wps 1 pin 12345670
ping 192.168.1.100
```

4.7.1.3 Peer device with DISPLAY method

1. Run the following commands on the development kit.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 15 6 6 US 300
wmiconfig --p2p autogo
```

2. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
p2p_peers
p2p_prov_disc 84:5d:d7:40:af:01 keypad
p2p_connect 84:5d:d7:40:af:01 12345670 display join
```

3. Run the following commands on the development kit.

```
wmiconfig --wps 1 pin 12345670
ping 192.168.1.100
```

4.8 Accept Invitation

4.8.1 Development kit accepting invitation from peer device

4.8.1.1 Development kit joining peer device using PUSH method

1. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_group_add
p2p_find
```

2. Run the following commands on the development kit.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 push
```

3. Run the following commands on the peer device.

```
p2p_peers
```

Open a new terminal, and go to directory of "/home/atheros/Atheros-P2P/"

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_invite group=wlan2 peer=84:5d:d7:40:af:01
wps_pbc 84:5d:d7:40:af:0
```

4. Run the following commands on the development kit.

```
wmiconfig --p2p join 12:03:7f:10:51:38 push
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
```

Logs on the development kit

```
[LOG]# Logs of foramtion
[LOG]Erri:sndfrm 44c88c
[LOG]Erri:sndfrm 44c86c
[LOG]Erri:sndfrm 44c8ac
[LOG]Erri:sndfrm 44c8cc
[LOG]Disconnected from DIRECT-7I
[LOG]noise_floor = -114
[LOG]Connected to DIRECT-7I,
[LOG]Erri:sndfrm 44c90c
[LOG]Erri:sndfrm 44c8cc
```

Logs on the peer device

```
# Logs of send invitation
[LOG]OK
# Start WPS PUSH on Group Interface of the peer device
[LOG]OK
# Logs of formation
[LOG]<2>CTRL-EVENT-EAP-STARTED 84:5d:d7:40:af:01
[LOG]<2>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
[LOG]<2>CTRL-EVENT-EAP-STARTED 84:5d:d7:40:af:01
[LOG]<2>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
[LOG]<2>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=14122 method=254
[LOG]<2>WPS-REG-SUCCESS 84:5d:d7:40:af:01 41424344-4546-4142-4344-454641424344
[LOG]<2>CTRL-EVENT-EAP-FAILURE 84:5d:d7:40:af:01
[LOG]<2>AP-STA-DISCONNECTED 84:5d:d7:40:af:01
[LOG]<2>AP-STA-CONNECTED 84:5d:d7:40:af:01
```

4.8.1.2 Development kit joining peer device using KEYPAD method

1. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_group_add
p2p_find
```

2. Run the following commands on the development kit.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
```

```

wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 display

```

3. Run the following commands on the peer device.

```
p2p_peers
```

Open a new terminal, and go to directory of "/home/atheros/Atheros-P2P/"

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_invite group=wlan2 peer=84:5d:d7:40:af:01
wps_pin 84:5d:d7:40:af:01 12345670

```

4. Run the following commands on the development kit.

```

wmiconfig --p2p join 12:03:7f:10:51:38 keypad 12345670
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10

```

4.8.1.3 Development kit joining peer device using DISPLAY method

1. Run the following commands on the peer device.

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_group_add
p2p_find

```

2. Run the following commands on the development kit.

```

wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 keypad

```

3. Run the following commands on the peer device.

```
p2p_peers
```

Open a new terminal, and go to directory of "/home/atheros/Atheros-P2P/"

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_invite group=wlan2 peer=84:5d:d7:40:af:01
wps_pin 84:5d:d7:40:af:01 12345670

```

4. Run the following commands on the development kit.

```

wmiconfig --p2p join 12:03:7f:10:51:38 display 12345670
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10

```

4.9 Re-invoke persistent group

4.9.1 Development kit re-invoke peer device

4.9.1.1 Development kit using PUSH method

1. Run the following commands on the peer device.


```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```

2. Run the following commands on the development kit.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 push
```

3. Run the following commands on the peer device.

```
p2p_connect 84:5d:d7:40:af:01 pbc auth persistent
```

4. Run the following commands on the development kit.

```
wmiconfig --p2p connect 02:03:7f:10:51:38 push persistent
wmiconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
wmiconfig --p2p cancel
```

5. Run the following commands on the peer device.

```
p2p_group_remove wlan2
p2p_find
```

6. Run the following commands on the development kit.

```
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p list_network
wmiconfig --p2p invite DIRECT-ju 02:03:7f:10:51:38 push persistent
ping 192.168.1.10
```

Logs on the development kit

```
# Logs of formation
[LOG]dev=0, go intend: 0
[LOG]shell> P2P GO neg result :
[LOG]Status      :SUCCESS
[LOG]P2P Role    :Client
[LOG]SSID        :DIRECT-ju
[LOG]Channel     :2437
[LOG]WPS Method  :PBC
[LOG]WPS connecting ...
[LOG]Erri:sndfrm 44c86c
[LOG]Erri:sndfrm 44c84c
[LOG]Disconnected from DIRECT-ju
[LOG]noise_floor = -111
[LOG]Connected to DIRECT-ju,
[LOG]Erri:sndfrm 44c84c
[LOG]Erri:sndfrm 44c86c
# Logs of p2p cancel
[LOG]Disconnected from DIRECT-ju
[LOG]noise_floor = -108
```

```
[LOG]shell> Erri:sndfrm 44c86c
# Logs of p2p list_network
[LOG]mac_addr[0] : 2:3:7f:10:51:38
[LOG]ssid[0] : DIRECT-ju
[LOG]mac_addr[1] : 0:0:0:0:0:0
[LOG]ssid[1] :
[LOG]mac_addr[2] : 0:0:0:0:0:0
[LOG]ssid[2] :
[LOG]mac_addr[3] : 0:0:0:0:0:0
[LOG]ssid[3] :
[LOG]mac_addr[4] : 0:0:0:0:0:0
[LOG]ssid[4] :
[LOG]mac_addr[5] : 0:0:0:0:0:0
[LOG]ssid[5] :
[LOG]mac_addr[6] : 0:0:0:0:0:0
[LOG]ssid[6] :
[LOG]mac_addr[7] : 0:0:0:0:0:0
[LOG]ssid[7] :
# Development kit send Reinvoke to the peer device using PUSH method
# Logs of reinvoke
[LOG]invit result : 0
[LOG]bssid : 12:3:7f:10:51:38
[LOG]Connected to DIRECT-ju,
[LOG]Erri:sndfrm 44c84c
[LOG]Erri:sndfrm 44c92c
```

Logs on the peer device

```
# Logs of formation
[LOG]<2>P2P-GO-NEG-SUCCESS
[LOG]<2>P2P-GROUP-FORMATION-SUCCESS
[LOG]<2>P2P-GROUP-STARTED wlan2 GO ssid="DIRECT-ju" freq=2437
passphrase="YjZKFO76" go_dev_addr=02:03:7f:10:51:38 [PERSISTENT]
# Logs of p2p_group_remove
[LOG]<2>P2P-GROUP-REMOVED wlan2 GO reason=REQUESTED
[LOG]> p2p_group_remove wlan2 OK
# Logs of being reinvoked
[LOG]<2>P2P-GROUP-STARTED wlan2 GO ssid="DIRECT-ju" freq=2437
passphrase="YjZKFO76" go_dev_addr=02:03:7f:10:51:38 [PERSISTENT]
```

4.9.1.2 Development kit using KEYPAD method

1. Run the following commands on the peer device.

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```

2. Run the following commands on the development kit.

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
```

- ```
wmicconfig --p2p nodelist
wmicconfig --p2p prov 02:03:7f:10:51:38 display
```
3. Run the following commands on the peer device.  

```
p2p_connect 84:5d:d7:40:af:01 12345670 display auth persistent
```
  4. Run the following commands on the development kit.  

```
wmicconfig --p2p connect 02:03:7f:10:51:38 keypad 12345670 persistent
wmicconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
wmicconfig --p2p cancel
```
  5. Run the following commands on the peer device.  

```
p2p_group_remove wlan2
p2p_find
```
  6. Run the following commands on the development kit.  

```
wmicconfig --p2p find
wmicconfig --p2p nodelist
wmicconfig --p2p list_network
wmicconfig --p2p invite DIRECT-ju 02:03:7f:10:51:38 keypad persistent
ping 192.168.1.10
```

#### 4.9.1.3 Development kit using DISPLAY method

1. Run the following commands on the peer device.  

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```
2. Run the following commands on the development kit.  

```
wmicconfig --p2p on
wmicconfig --p2p setconfig 0 6 6 US 300
wmicconfig --p2p find
wmicconfig --p2p nodelist
wmicconfig --p2p prov 02:03:7f:10:51:38 keypad
```
3. Run the following commands on the peer device.  

```
p2p_connect 84:5d:d7:40:af:01 12345670 keypad auth persistent
```
4. Run the following commands on the development kit.  

```
wmicconfig --p2p connect 02:03:7f:10:51:38 display 12345670 persistent
wmicconfig --ipstatic 192.168.1.100 255.255.255.0 192.168.1.10
ping 192.168.1.10
wmicconfig --p2p cancel
```
5. Run the following commands on the peer device.  

```
p2p_group_remove wlan2
p2p_find
```
6. Run the following commands on the development kit.  

```
wmicconfig --p2p find
wmicconfig --p2p nodelist
wmicconfig --p2p list_network
```

```

wmiconfig --p2p invite DIRECT-ju 02:03:7f:10:51:38 display persistent
ping 192.168.1.10

```

## 4.9.2 Peer device re-invoke the development kit

### 4.9.2.1 Development kit using PUSH method

1. Run the following commands on the peer device.  

```

root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find

```
2. Run the following commands on the development kit.  

```

wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 push

```
3. Run the following commands on the peer device.  

```

p2p_connect 84:5d:d7:40:af:01 pbc auth persistent

```
4. Run the following commands on the development kit.  

```

wmiconfig --p2p connect 02:03:7f:10:51:38 push persistent
ping 192.168.1.100
wmiconfig --p2p cancel

```
5. Run the following commands on the peer device.  

```

p2p_group_remove wlan2

```
6. Run the following commands on the development kit.  

```

wmiconfig --p2p find

```
7. Run the following commands on the peer device.  

```

p2p_find
p2p_peers
list_network
p2p_invite persistent=0 group=wlan2 peer=84:5d:d7:40:af:01

```
8. Run the following commands on the development kit.  

```

wmiconfig -p2p invite_auth 02:03:7f:10:51:38
wmiconfig --p2p start_pers DIRECT-bF (called only if DUT is GO in
persistent group)

```
9. Ping peer device again.

### Logs on the development kit

```

Logs of formation
[LOG]dev=0, go intend: 15
[LOG]shell> P2P GO neg result :
[LOG]Status :SUCCESS
[LOG]P2P Role :GO
[LOG]SSID :DIRECT-bF

```

```

[LOG]Channel :2437
[LOG]WPS Method :PBC
[LOG]WPS connecting ...Connected to DIRECT-bF,
[LOG]Erri:sndfrm 44c90c
[LOG]Disconnected from DIRECT-bF
[LOG]noise_floor = -115
[LOG]Erri:sndfrm 44c96c
[LOG]Connected to DIRECT-bF,
[LOG]Erri:sndfrm 44c88c
[LOG]Error in starting the task ...
[LOG]Error in starting the task ...
Logs of release DHCP to the peer device automatically
[LOG]ip status 6401a8c0 1
Logs of p2p cancel
[LOG]dhcppool cleaned.Disconnected from DIRECT-bF
[LOG]noise_floor = -109
[LOG]Disconnected from DIRECT-bF
[LOG]noise_floor = -109
Logs of being reinvoked
[LOG]invitation req received from : 84:5d:d7:40:af:1
[LOG]invite result : 0
[LOG]sa : 2:3:7f:10:51:38
[LOG]Connected to DIRECT-bF,
[LOG]Error in starting the task ...
[LOG]shell> Connected to DIRECT-bF,
[LOG]Error in starting the task ...
[LOG]Error in starting the task ...
[LOG]ip status 6401a8c0 1

```

## Logs on peer device

```

Logs of formation
[LOG]<2>P2P-GO-NEG-SUCCESS
[LOG]<2>P2P-GROUP-FORMATION-SUCCESS
[LOG]<2>P2P-GROUP-STARTED wlan2 GO ssid="DIRECT-ju" freq=2437
passphrase="YjZKF076" go_dev_addr=02:03:7f:10:51:38 [PERSISTENT]
Logs of p2p_group_remove
[LOG]<2>P2P-GROUP-REMOVED wlan2 GO reason=REQUESTED
[LOG]> p2p_group_remove wlan2 OK
Logs of list_network
[LOG]network id / ssid / bssid / flags
[LOG]0 DIRECT-bF 84:5d:d7:40:af:01 [DISABLED] [P2P-
PERSISTENT]
Logs of reinvoke
[LOG]OK
[LOG]<2>P2P-INVITATION-RESULT status=0
[LOG]<2>P2P-GROUP-STARTED wlan2 client ssid="DIRECT-bF" freq=0
passphrase="fkZeVs0l" go_dev_addr=84:5d:d7:40:af:01 [PERSISTENT]

```

### 4.9.2.2 Development kit using KEYPAD method

1. Run the following commands on the peer device.  

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```
2. Run the following commands on the development kit.  

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 display
```
3. Run the following commands on the peer device.  

```
p2p_connect 84:5d:d7:40:af:01 12345670 display auth persistent
```
4. Run the following commands on the development kit.  

```
wmiconfig --p2p connect 02:03:7f:10:51:38 keypad 12345670 persistent
ping 192.168.1.100
wmiconfig --p2p cancel
```
5. Run the following commands on the peer device.  

```
p2p_group_remove wlan2
```
6. Run the following commands on the development kit.  

```
wmiconfig --p2p find
```
7. Run the following commands on the peer device.  

```
p2p_find
p2p_peers
list_network
p2p_invite persistent=0 group=wlan2 peer=84:5d:d7:40:af:01
```
8. Run the following commands on the development kit.  

```
wmiconfig -p2p invite_auth 02:03:7f:10:51:38
wmiconfig --p2p start_pers DIRECT-bF (called only if DUT is GO in
persistent group)
```
9. Ping peer device again.

### 4.9.2.3 Development kit using DISPLAY method

1. Run the following commands on the peer device.  

```
root@Atheros-P2P:/home/atheros/Atheros-P2P# ./wpa_cli
p2p_find
```
2. Run the following commands on the development kit.  

```
wmiconfig --p2p on
wmiconfig --p2p setconfig 0 6 6 US 300
wmiconfig --p2p find
wmiconfig --p2p nodelist
wmiconfig --p2p prov 02:03:7f:10:51:38 keypad
```
3. Run the following commands on the peer device.

- ```
p2p_connect 84:5d:d7:40:af:01 12345670 display auth persistent
```
4. Run the following commands on the development kit.

```
wmiconfig --p2p connect 02:03:7f:10:51:38 keypad 12345670 persistent  
ping 192.168.1.100  
wmiconfig --p2p cancel
```
 5. Run the following commands on the peer device.

```
p2p_group_remove wlan2
```
 6. Run the following commands on the development kit.

```
wmiconfig --p2p find
```
 7. Run the following commands on the peer device.

```
p2p_find  
p2p_peers  
list_network  
p2p_invite persistent=0 group=wlan2 peer=84:5d:d7:40:af:01
```
 8. Run the following commands on the development kit.

```
wmiconfig -p2p invite_auth 02:03:7f:10:51:38  
wmiconfig --p2p start_pers DIRECT-bF (called only if DUT is GO in  
persistent group)
```
 9. Ping peer device again.

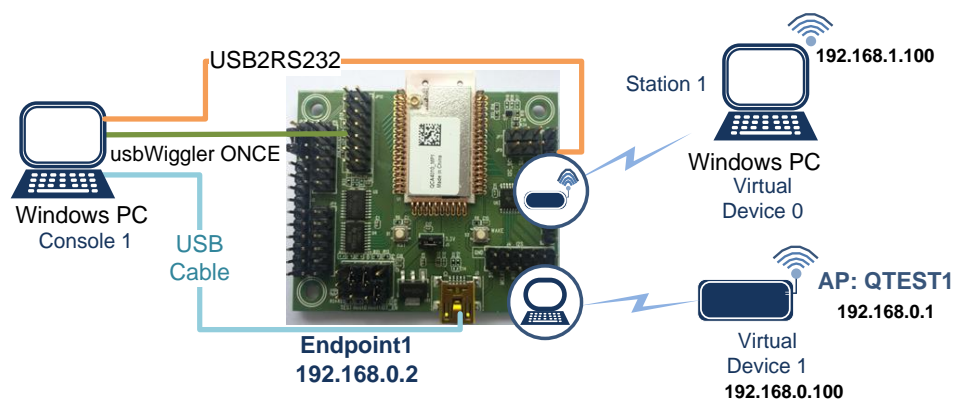
5 Concurrency Demo

The concurrency demo enables two virtual devices Device 0 and Device 1 on the development kit. Device 0 can operate as AP/P2P device/P2P Client/P2P Go mode while Device 1 can operate as Station. The concurrency demo supports both Single Channel Concurrency (SCC) where Device 1 and Device 0 operate on the same channel, and Multiple Channel Concurrency (MCC) where Device 1 and Device 0 operate on different channels.

5.1 Setup

This section describes how to set up the environment for testing SCC/MCC concurrency. See section 3.3 and 4.1 for shell and P2P commands.

1. Build and download the MCC demo image (MCC image supports SCC implicitly) to the development kit. The following images are available in the release package.
 - raw_flashimage_AR401X_REV6_IOT_MP1_hostless_ap+sta_mcc_singleband_1bitflash.bin
2. Connect Windows PC (Console 1) to the development kit using a USB cable. Start a console application to input commands.
3. Set up an external AP device and make sure the development kit and the AP are within the same subnet. Following is a configuration example.



4. On the development kit, start Virtual Device 0 (SoftAP).

```
wmiconfig --device 0
wmiconfig --channel 6
wmiconfig --p 12345678
```



```
wmiconfig --wpa 2 CCMP CCMP
wmiconfig --mode ap
wmiconfig --connect ap-test
```

5. On Station 1, connect to the Virtual Device 0 (SoftAP).

Station 1 is assigned IP address 192.168.1.100.

6. On Station 1, ping Virtual Device 0 (SoftAP).

```
Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.10: bytes:32 time<1ms TTL=64
Reply from 192.168.1.10: bytes:32 time<1ms TTL=64
Reply from 192.168.1.10: bytes:32 time<1ms TTL=64
Reply from 192.168.1.10: bytes:32 time<1ms TTL=64
```

```
Ping statistics for 192.168.1.10:
    Packets: Sent = 4
Approximate round trip times in milli-seconds:
    Minimum: 0ms. Maximum: 0ms. Average: 0ms.
```

7. On the development kit, connect Virtual Device 1 (station) to the external AP.

```
wmiconfig --device 1
wmiconfig --channel 1
wmiconfig --wpa 2 CCMP CCMP
wmiconfig --p 12345678
wmiconfig --connect ap-external
wmiconfig --ipdhcp
```

Virtual Device 1 is assigned IP address 192.168.0.100.

8. On Virtual Device 1, ping the external AP.

```
ping 192.168.0.1
Ping reply from 192.168.0.1: time<1ms
```

6 Power Measurement

6.1 Overview

This chapter describes the procedures to measure the current consumption of Wi-Fi module (RB02) in various operation modes on the development platform. The following topics are covered in this document:

- Suspend mode current measurement
- Sleep mode current measurement
- DTIM mode current measurement
- Connection idle mode current measurement
- TCP Tx/Rx current measurement
- Green Tx current measurement (UDP)
- LPL current measurement (UDP)
- WLAN-BLE Coex Scenarios current measurement

6.1.1 Test environment

Power consumption is measured in the following environment setting:

Table 6-1 Power measurement environment setup

Hardware	RB01-010: 20-YA162-H1 RB02-010: 20-YA163-H1
Firmware	5.0
Environment	Room temperature, regulated 3.3 V supply unless otherwise specified. Measurement is taken within shield box in shield Room.
Test instrument	Agilent N6705B DC1 power analyzer mainframe for dynamic measurements, with four-wire sensing disabled. Channel 1 is used to measure the current from the 3.3 V supply to the development kit.

6.1.2 Power test preparations

6.1.2.1 Hardware preparation for RB01/RB02 development platform

The hardware rework for RB01/RB02 is quite simple. Just need to remove R34 which is the power input of the onboard sensor. Avoid using heat gun for the rework as it might influence the sensor electronic characteristics.



Figure 6-1 Hardware rework of RB01/RB02 for power measurement

Set Host0 & IOT as “1” and Host1 as “0”. Remove jumper of TEST as shown in [Figure 6-2](#).

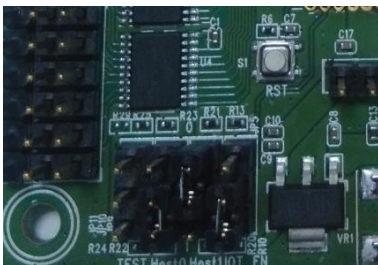


Figure 6-2 Function jumper configuration of RB01 for power measurement

6.1.3 Power measurement points

6.1.3.1 Testing points for RB01/RB02 development platform

Configure N6705B's channel 1 as “2 Quadrant Power Supply” mode. N6705B supplies external 3.3V voltage to RB02 board, and measures current value. Remove the original jumper at J5 to break power connection between baseboard and module.

NOTE: The mini-USB connector should be connected to supply extra power for peripheral interface on baseboard.

- **J5:** Pin2 connects to the positive pole of the power analyzer.
- **J4:** GND pin connects to the negative pole of the power analyzer.

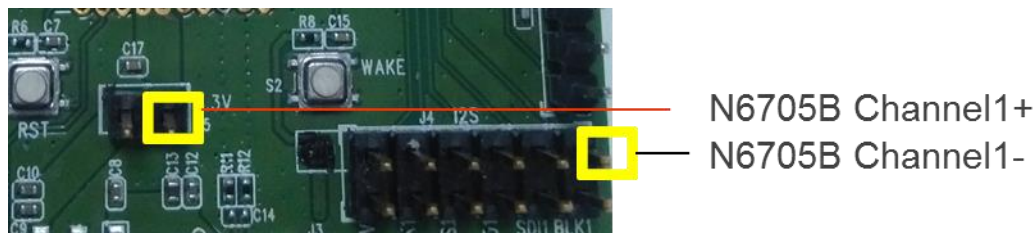


Figure 6-3 Power measurement points for RB01/RB02

6.1.4 Console connection

J1 over RB01 provide debug UART connection with the RB02 module.

1. Use a USB-UART cable to connect the RB01/RB02 kit to a Windows-based PC through JP21. Otherwise, install TB39-051 over JP21 and use USB-RS232 cable to connect the kit.

For the RB01/RB02 kit, you can only use USB-UART cable. PIN6 to RXD, PIN7 to GND and PIN8 for TXD of USB-UART cable.

2. Start a terminal application from the PC and select the lower port to connect using the port setting **115200, 8, n, 1, no flow control**.
3. The serial terminal application displays:

```
shell >
```

The console connection is established. Type the following command at the terminal to view the version information. The driver and the firmware versions can be different.

```
wmiconfig --version
```

6.2 Measuring suspend current

In Suspend mode, all circuits except the Wakeup Manager and PMU circuits are shut down.

Command

- `wmiconfig --suspend`
 - Enable store-recall function
- `wmiconfig --suspend`
 - Power down the QCA4010/QCA4012 chip through the CHIP_PWD signal.

Test procedures

1. Connect the board to power analyzer as described in section [6.1.3](#).
2. Set up console connection with a PC, see section [6.1.4](#).
3. Enter the following commands at the console:

```
wmiconfig --pwrmode 0
wmiconfig --suspend
wmiconfig --connect <Ref_AP_SSID>
wmiconfig --suspend <time in ms>
```
4. Observe the analyzer and take the measurements.

6.3 Measuring sleep current

Test procedure

1. Connect the board to power analyzer as described in section [6.1.3](#).

2. Set up console connection with a PC, see section [6.1.4](#).
3. Configure the reference AP to 2.4G mode and set DTIM to 1.
4. Enter the following commands at the console:

```
wmiconfig --driver down
wmiconfig --driver up
wmiconfig --pmparms --dp 3
wmiconfig --pwrmode 1
wmiconfig --scantctrl 0 0
wmiconfig --connect <Ref_AP_SSID>
sleep <time in second>
```
5. Observe the analyzer and measure power between beacons. Take the current measurements for sleep duration.

6.4 Measuring DTIM time-averaged current

6.4.1 Measuring DTIM1 current

Test procedure

1. Connect the board to power analyzer as described in section [6.1.3](#).
2. Set up console connection with a PC, see section [6.1.4](#).
3. Configure the reference AP to 2.4G mode and set DTIM to 1.
4. Enter the following commands at the console:

```
wmiconfig --driver down
wmiconfig --driver up
wmiconfig --pmparms --dp 3
wmiconfig --pwrmode 1
wmiconfig --scantctrl 0 0
wmiconfig --connect <Ref_AP_SSID>
sleep <time in second>
```
5. Observe the analyzer and take the measurements for DTIM1 duration.

6.4.2 Measuring DTIM3 current

Test procedure

1. Connect the board to power analyzer as described in section [6.1.3](#).
2. Set up console connection with a PC, see section [6.1.4](#).
3. Configure the reference AP to 2.4G mode and set DTIM to 3.
4. Enter the following commands at the console:

```
wmiconfig --driver down
wmiconfig --driver up
wmiconfig --pmparms --dp 3
```

```
wmiconfig --pwrmode 1
wmiconfig --scancntrl 0 0
wmiconfig --connect <Ref_AP_SSID>
sleep <time in second>
```

5. Observe the analyzer and take the measurements for DTIM3 duration.

6.4.3 Measuring DTIM5 current

Test procedure

1. Connect the board to power analyzer as described in section [6.1.3](#).
2. Set up console connection with a PC, see section [6.1.4](#).
3. Configure the reference AP to 2.4G mode and set DTIM to 5.
4. Enter the following commands at the console:

```
wmiconfig --driver down
wmiconfig --driver up
wmiconfig --pmparms --dp 3
wmiconfig --pwrmode 1
wmiconfig --scancntrl 0 0
wmiconfig --connect <Ref_AP_SSID>
sleep <time in second>
```

5. Observe the analyzer and take the measurements for DTIM5 duration.

6.4.4 Measuring DTIM10 current

Test procedure

1. Connect the board to power analyzer as described in section [6.1.3](#).
2. Set up console connection with a PC, see section [6.1.4](#).
3. Configure the reference AP to 2.4G mode and set DTIM to 10.
4. Enter the following commands at the console:

```
wmiconfig --driver down
wmiconfig --driver up
wmiconfig --pmparms --dp 3
wmiconfig --pwrmode 1
wmiconfig --scancntrl 0 0
wmiconfig --connect <Ref_AP_SSID>
sleep <time in second>
```

5. Observe the analyzer and take the measurements for DTIM10 duration.

6.5 Measuring MAX Perf connection idle current

Test procedure

1. Connect the board to power analyzer as described in section 6.1.3.
2. Set up console connection with a PC, see section 6.1.4.
3. Configure the reference AP to 2.4G mode.
4. Enter the following commands at the console:

```
wmiconfig --pwrmode 0//no power save  
wmiconfig --connect <Ref_AP_SSID>
```
5. Observe the analyzer and take the measurements for connection idle duration.

6.6 Measuring TCP Tx/Rx current

Test procedures

1. Connect the board to power analyzer as described in section 6.1.3.
2. Set up console connection with a PC, see section 6.1.4.
3. Configure the reference AP to 2.4G mode.
4. Enter the following commands at the debug console:

```
wmiconfig --pwrmode 1  
wmiconfig --connect <Ref_AP_SSID>
```
5. For TCP Tx current measurement:
 - a. Enter the following command at the shell:

```
ath_console rx <Console_IPADDR> 7008 tcp v4
```
 - b. Enter the following command at the console:

```
benchttx <Console_IPADDR> 7008 tcp 1400 0 30 0
```
6. For TCP Rx current measurement:
 - a. Enter the following command at the console:

```
benchrx tcp 7008
```
 - b. Enter the following command at the shell:

```
ath_console tx <DUT_IPADDR> 7007 tcp 1400 0 30 0 v4
```
7. Observe the analyzer and take the measurements for TCP Tx/Rx duration.

6.7 Measuring Green Tx current (UDP uplink)

The Green Tx feature allows the device to save power when communicating with a nearby station or access point, where high output power is not required to sustain reliable communications. In such cases, the transmitter can reduce the transmit power to obtain current saving, and at the same time maintaining high uplink throughput.

NOTE: The Green Tx enabling/disabling command can be used only once. Re-issuing the command cannot change the Green Tx status. To change the Green Tx status again, reset the board.

Test procedures

1. Connect the board to power analyzer as described in section 6.1.3.
2. Set up console connection with a PC, see section 6.1.4.
3. Configure the reference AP to 2.4G mode.
4. Enter the following commands in the debug console:

```
wmiconfig --pwrmode 0
wmiconfig --setrate 1 7
wmiconfig --connect <Ref_AP_SSID>
wmiconfig --gtx <0|1> //disable or enable Green Tx, used only once
```
5. Enter the following command at the remote console:

```
ath_console rx <Console_IPADDR> 7007 udp v4
```
6. Enter the following command at the console to start the benchmark test:

```
benchtx <Console_IPADDR> 7007 udp 1400 0 60 0
```
7. Observe the analyzer and take the measurements for Green Tx duration.

6.8 Measuring LPL current (UDP downlink)

Test procedures

1. Connect the board to power analyzer as described in section 6.1.3.
2. Set up console connection with a PC, see section 6.1.4.
3. Configure the reference AP to 2.4G mode.
4. Enter the following commands in the debug console:

```
wmiconfig --pwrmode 0
wmiconfig --connect <Ref_AP_SSID>
wmiconfig --lpl <0|1> //disable or enable lpl, used only once
```
5. Enter the following command at the remote console:

```
ath_console tx <DUT_IPADDR> 7008 udp 1400 0 60 0 v4
```
6. Enter the following command at the console to start the benchmark test:

```
benchrx udp 7008
```
7. Observe the analyzer and take the measurements for LPL duration.

7 APIs

This section is a reference for application developers. The QCA4010/QCA4012 hostless SDK run time model and the APIs are described in this document.

7.1 System overview

The QCA4010/QCA4012 SoC provides both 802.11 Wireless LAN (WLAN) connectivity as well as a base programming environment to run applications on. The WLAN subsystem includes:

- Wireless media access control (MAC)
- Radio
- Baseband
- IEEE 802.11 protocol processing handled by an on-chip network processor

The non-WLAN portion of the SoC consists of:

- Tensilica Xtensa processor, up to 130 MHz
- 1MBytes ROM, 1.4MBytes RAM
- 1024 Bytes One-time programmable (OTP) memory
- Internal Power Management Unit (PMU) with switch and linear regulators
- A master SPI controller that also supports quad mode SPI flash
- 2 UART interfaces
- GPIOs, I²C, and I²S
- PWM
- ADC
- HW crypto
- JTAG interface

The QCA4010/QCA4012 SoC can be used in Hosted mode with external MCU or in fully self-contained Hostless mode where all code runs on the internal Tensilica processor. The current SDK release supports the Wi-Fi module in hostless configuration only.

7.2 Hostless SDK

The hostless SDK enables the QCA4010/QCA4012 SoC to process IP packets in addition to Wi-Fi connection management. The hostless software model consists of two domains:

- Target domain – Running the WLAN sub-system, the internal chip support code, and ThreadX framework.
- PseudoHost domain

The Target software handles the underlying hardware and wireless protocols, and directly manages the WLAN hardware. The Target software is provided by Qualcomm Technologies with optional tweaks for customization.

The PseudoHost domain software consists of the high-level application code, user services, and the libraries provided by Qualcomm Technologies to interface to the Target domain. The PseudoHost domain software is mostly customized by vendors leveraging the libraries supplied by Qualcomm Technologies.

The Target and the PseudoHost domains share the same CPU and physical memory resources, thus some general software attributes are used to distinguish the Target and the PseudoHost domains.

7.2.1 Target domain

The Target domain mostly operates in one of the following ways:

- **Interrupt Service Routine (ISR)** – the entry point to add hardware or software-generated stimulus to inject new events on the processor
- **Delayed Service Routine (DSR)** – executing most of the useful work within the Target domain.
- **Background Routine (BGR)** – the entry point to schedule the PseudoHost domain requests
- **ThreadX thread** – where the PseudoHost domain applications and the associated Target side requests are serviced, in addition to some of the Qualcomm Technologies supplied Target functionality

The core scheduler resolves the operations for each software type following the priorities from high to low:

ISR → High priority BGR → DSR → Low priority BGR → ThreadX thread

ISRs are not scheduled and are executed immediately, preempting any non-ISR code. After ISR execution is complete, the software resumes where it was interrupted. ISR itself may add new tasks that get scheduled to run. DSRs and BGRs never preempt an executing ThreadX thread. A scheduling decision is made when a BGR/DSR completes or when a ThreadX thread completes or blocks. Scheduling across ThreadX threads is handled with normal ThreadX prioritization.

7.2.2 PseudoHost domain

The PseudoHost domain runs on the ThreadX framework, matching the ThreadX of the Target domain. The QCA4010/QCA4012 hostless SDK supports a single instance of ThreadX RTOS (a single scheduler) shared between the Target and the PseudoHost domains. Customer PseudoHost software can create multiple threads using small thread stacks for customer applications. The APIs supplied by Qualcomm Technologies restrict the stack requirement to a few hundred (for example, 512) Bytes.

The PseudoHost domain software interfaces to the Target resources in either of the following ways:

- Using the Wireless Networking APIs (WNAPIs) provided by Qualcomm Technologies to issue requests to the Target domain
- Using the APIs provided by Qualcomm Technologies to directly access the Target resources

7.2.3 System memory map

The QCA4010/QCA4012 system memory includes the following components:

- ROM: 1Mbytes (0x0090 0000 – 0x009F FFFF)
- RAM
 - 576 Kbytes dedicated Instruction RAM, starting from 0x00A0 0000
 - 384 Kbytes dedicated Data RAM, starting from 0x0042 0000
 - 512 Kbytes configurable RAM bank (in 32-Kbyte blocks) for either data or instruction

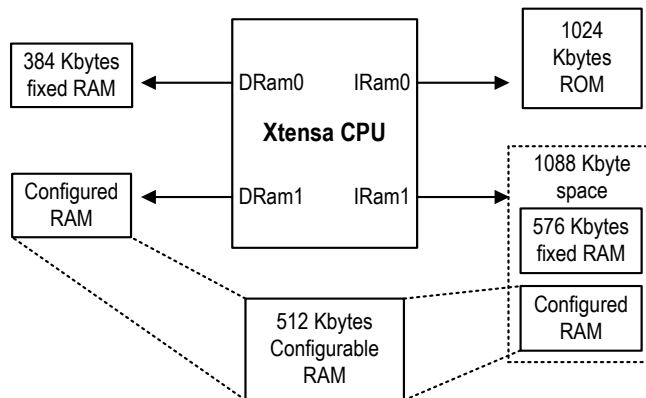


Figure 7-1 System memory allocation

The instruction RAM starts at 0x00A0 0000. The end address, depending on the number of RAM blocks allocated for instruction storage, varies from 0x00A8 FFFF (no blocks allocated) to 0x00B0 FFFF (all 16 blocks allocated).

The data RAM starts at 0x0042 0000. The end address, depending on the number of RAM blocks allocated for data storage, varies from 0x0047 FFFF (no blocks allocated) to 0x004F FFFF (all 16 blocks allocated).

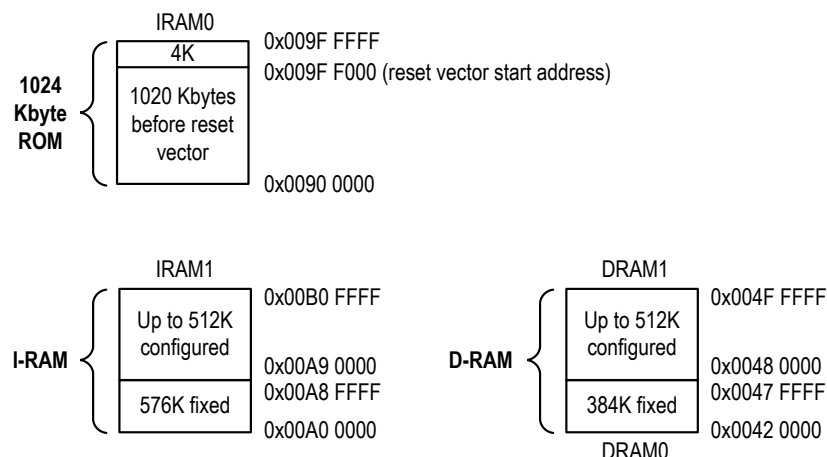


Figure 7-2 Address allocation for ROM and RAM

Determination of IRAM/DRAM usage

By default, all of the configurable RAM (512Kbytes) is configured as DRAM unless customer changes the **NUM_IRAM_BANKS** parameter in the tunable file.

Use `xt-objdump` command to get IRAM consumption.

Example

```
xt-objdump -xww iot_demo.out
```

Where **iot_demo.out** is a sample application provided in QCA4010/QCA4012 SDK. A customer can do the same with their application's .out file.

Sections

Idx	Name	Size	VMA	LMA	File off	Align	Flags
0	.lit4	00007100	009f9000	009f9000	00054553	2**0	CONTENTS
1	.literals	0000384c	00a00100	00a00100	00000094	2**2	CONTENTS, ALLOC, LOAD, READONLY, DATA
2	.rodata	00010020	00446000	00446000	000038e0	2**4	CONTENTS, ALLOC, LOAD, READONLY, DATA
3	.data	00001950	00456020	00456020	00013900	2**4	CONTENTS, ALLOC, LOAD, DATA
4	.bss	00004238	00457970	00457970	00015250	2**4	ALLOC
5	.text	0003f303	00a05000	00a05000	00015250	2**4	CONTENTS, ALLOC, LOAD, READONLY, CODE

Add text + literals + rodata from above output to get IRAM consumption, that is, $3f303 + 384c + 10020 = 52B6F$, converting to decimal $338799/1024 = 330K$.

DRAM consumption depends on how many dynamic memory allocations are being made. Use **qcom_free_heap_size_get()** API to see how much heap size is available. With default demo app (iot_demo) you can use `wmiconfig` command to get available DRAM space.

Example

```
shell> wmiconfig
SSID           = mk-ntgr
Phy Mode       = mixed
Power Mode     = Power Save
Mac Addr       = 74:df:bf:ba:fc:33
Mode           = Station
Channel        = 1
Heapsize       = 457600
Buffer : Mac (8+0), HTC (3+0), FW (1+0), Free 8, tx (alloc 1, free 0)
```

Where **Heapsize** indicates available DRAM space when running `iot_demo` app.

7.3 Network APIs

The QCA4010/QCA4012 SDK exposes a set of APIs to application developers for creating TCP and UDP sockets and sending/receiving data over these sockets.

7.3.1 qcom_ipconfig

Definition	Set/get IPv4 parameters, or trigger DHCP client.	
Prototype	A_STATUS qcom_ipconfig (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	A_INT32 mode,	Command mode. Possible values- IP_CONFIG_QUERY (0) – get IP v4 parameters IP_CONFIG_SET (1) – assign static IP v4 parameters IP_CONFIG_DHCP (2) – Run DHCP client
	A_UINT32 *address,	IPv4 address
	A_UINT32 *submask,	IPv4 subnet mask
	A_UINT32 *gateway,	IPv4 gateway address
)	
Return value	A_OK / A_ERROR	

7.3.2 qcom_ip6_address_get

Definition	Get IPv6 parameters.	
Prototype	A_STATUS qcom_ip6_address_get (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT8 *v6Global,	IPv6 global address
	A_UINT8 *v6Link,	IPv6 local address
	A_UINT8 *v6DefGw,	IPv6 default gateway
	A_UINT8 *v6GlobalExtd,	IPv6 extended global address
	A_INT32 *LinkPrefix,	IPv6 link prefix
	A_INT32 *GlbPrefix,	IPv6 global prefix
	A_INT32 *DefGwPrefix	IPv6 default gateway prefix
	A_INT32 *GlbPrefixExtd	IPv6 extended global prefix
)	
Return value	A_OK / A_ERROR	

7.3.3 qcom_ping

Definition	Send an IPv4 ping. The Host is blocked until a result is received from the Target.	
Prototype	A_STATUS qcom_ping (
	A_UINT32 host,	IPv4 destination address
	A_UINT32 size,	Size of ping payload in bytes
)	
Return value	A_OK / A_ERROR	

7.3.4 qcom_ping6

Definition	Send an IPv6 ping. The Host is blocked until a result is received from the Target.	
Prototype	A_STATUS qcom_ping6 (
	A_UINT8* host,	IPv6 destination address
	A_UINT32 size,	Size of ping payload in bytes
)	
Return value	A_OK / A_ERROR	

7.3.5 qcom_ip6config_router_prefix

Definition	Set the IPv6 router prefix (SoftAP mode only)	
Prototype	A_STATUS qcom_ip6config_router_prefix (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	A_UINT8* addr,	Prefix
	A_INT32 prefix_length,	Length of prefix
	A_INT32 prefix_lifetime,	Lifetime of prefix
	A_INT32 valid_lifetime	
)	
Return value	A_OK	

7.3.6 qcom_bridge_mode_enable

Definition	Enable/Disable bridging	
Prototype	A_STATUS qcom_bridge_mode_enable (
	A_UINT16 bridgemode	1 = Enable; 0 = Disable
)	
Return value	A_OK	

7.3.7 qcom_set_hostname

Definition	Set DHCP hostname.	
Prototype	A_STATUS qcom_set_hostname(
	A_UINT8 device_id,	Interface ID
	A_CHAR *pstr	DHCP hostname
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.8 qcom_dhcps_set_pool

Definition	Set DHCP pool parameters.	
Prototype	A_STATUS qcom_dhcps_set_pool(
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 startip,	The first IP address of DHCP server IP pool

	<i>A_UINT32 endip,</i>	The last IP address of DHCP server IP pool
	<i>A_UINT32 leasetime,</i>	Lease time
)	
Return value	A_OK / A_ERROR	

7.3.9 qcom_dhcps_release_pool

Definition	Release DHCP address.	
Prototype	A_STATUS qcom_dhcps_release_pool(
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1 (default 0)
)	
Return value	A_OK / A_ERROR	

7.3.10 qcom_dns_server_address_get

Definition	Get DNS address from DHCP result.	
Prototype	A_STATUS qcom_dns_server_address_get (
	<i>A_UINT32 *pdns</i>	Pointer to array where the DNS addresses are to be copied. Caller must ensure that the array size is equal to or greater than 3.
	<i>A_UINT32* number</i>	Number of addresses
)	
Return value	A_OK / A_ERROR	

7.3.11 qcom_dnss_enable

Definition	Enable or disable DNS server	
Prototype	A_STATUS qcom_dnss_enable(
	<i>A_BOOL enable</i>	1 = Enable; 0 = Disable
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.12 qcom_dns_local_domain

Definition	Set DNS local domain name.	
Prototype	A_STATUS qcom_dns_local_domain(
	<i>A_CHAR *local_domain</i>	Local domain name
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.13 qcom_dnsc_enable

Definition	Enable/Disable DNS client function.	
Prototype	A_STATUS qcom_dnsc_enable (
	A_BOOL enable	1 = Enable, 0 = Disable
)	
Return value	A_OK / A_ERROR	

7.3.14 qcom_dnsc_add_server_address

Definition	Add DNS server IP address.	
Prototype	A_STATUS qcom_dnsc_add_server_address (
	A_UINT8* ipaddress	IP address of DNS server
	A_UINT8 type	Type = AF_INET (IPv4) = AF_INET6 (IPv6)
)	
Return value	A_OK / A_ERROR	

7.3.15 qcom_dnsc_del_server_address

Definition	Delete DNS server IP address.	
Prototype	A_STATUS qcom_dnsc_del_server_address (
	A_UINT8 *ipaddress	IP address of DNS server
	A_UINT8 type	Type = AF_INET (IPv4) = AF_INET6 (IPv6)
)	
Return value	A_OK / A_ERROR	

7.3.16 qcom_dnsc_get_host_by_name

Definition	Get IP address by URL.	
Prototype	A_STATUS qcom_dnsc_get_host_by_name (
	A_CHAR *pname,	A pointer to URL
	A_UINT32 *pipaddress,	A pointer to IP address
)	
Return value	A_OK / A_ERROR	

7.3.17 qcom_dns_entry_create

Definition	Create a DNS server entry (IPv4).	
Prototype	A_STATUS qcom_dns_entry_create (
	A_CHAR *hostname,	Domain name pointer
	A_UINT32 ipaddress,	IPv4 address

)	
Return value	A_OK / A_ERROR	

7.3.18 qcom_dns_entry_delete

Definition	Delete a DNS server entry (IPv4).	
Prototype	A_STATUS qcom_dns_entry_delete (
	A_CHAR *hostname,	Domain name pointer
	A_UINT32 ipaddress,	IPv4 address
)	
Return value	A_OK	

7.3.19 qcom_dns_6entry_create

Definition	Create a DNS server entry (IPv6).	
Prototype	A_STATUS qcom_dns_6entry_create (
	A_CHAR *hostname,	Domain name pointer
	A_UINT8 *ipaddress,	IPv6 address
)	
Return value	A_OK	

7.3.20 qcom_dns_6entry_delete

Definition	Delete a DNS server entry (IPv6).	
Prototype	A_STATUS qcom_dns_6entry_delete (
	A_CHAR *hostname,	Domain name pointer
	A_UINT8 *ipaddress,	IPv6 address
)	
Return value	A_OK / A_ERROR	

7.3.21 qcom_dnsc_get_host_by_name2

Definition	Get IP address by URL.	
Prototype	A_STATUS qcom_dnsc_get_host_by_name2 (
	A_CHAR *pname,	A pointer to URL
	A_UINT32 *pipaddress,	A pointer to IP address
	A_INT32 domain,	2 = IPv4; 3 = IPv6
	A_UINT32 mode	1 = gETHOSTbyname2; 2 = resolvehostname

)	
Return value	A_OK / A_ERROR	

7.3.22 qcom_sntp_srvr_addr

Prototype	void qcom_sntp_srvr_addr(
	<i>A_INT32 flag,</i>	Flag to add/delete the address 1 = Add server address 2 = Delete server address
	<i>char *srvr_addr</i>	Pointer to SNTP Server address
)	
Description	Configure SNTP Server address.	
Return value	0 on success, -1 on failure.	

7.3.23 qcom_sntp_get_time

Prototype	void qcom_sntp_get_time(
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1 (default 0)
	<i>tSntpTime *time</i>	Pointer to SNTP time structure
)	
Description	Get SNTP time. The time parameter contains the current date time information.	
Return value	0 on success, -1 on failure.	

7.3.24 qcom_sntp_get_time_of_day

Prototype	void qcom_sntp_get_time_of_day(
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1 (default 0)
	<i>tSntpTM *time</i>	Pointer to SNTP structure that contains time in seconds and milliseconds
)	
Description	Get SNTP time of day in seconds and milliseconds.	
Return value	0 on success, -1 on failure.	

7.3.25 qcom_sntp_zone

Prototype	void qcom_sntp_zone(
	<i>A_INT32 hour,</i>	Time stored in hours
	<i>A_INT32 min,</i>	Time stored in minutes
	<i>A_BOOL add_sub,</i>	Time to add or subtract to get the appropriate time zone
	<i>A_BOOL enable</i>	Enable or disable day light saving
)	
Description	Configure SNTP time zone and enable/disable day light saving.	

Prototype	void qcom_sntp_zone(
	<i>A_INT32 hour,</i>	Time stored in hours
	<i>A_INT32 min,</i>	Time stored in minutes
	<i>A_BOOL add_sub,</i>	Time to add or subtract to get the appropriate time zone
	<i>A_BOOL enable</i>	Enable or disable day light saving
)	
Return value	0 on success, -1 on failure.	

7.3.26 qcom_sntp_query_srvr_address

Prototype	void qcom_sntp_query_srvr_address(
	<i>A_UINT8 device_id,</i>	Unused (default 0)
	<i>SNTP_QUERY_SRVR_ADDRESS *addr</i>	Pointer to SNTP server address structure
)	
Description	Query the SNTP address.	
Return value	0 on success, -1 on failure.	

7.3.27 qcom_enable_sntp_client

Prototype	void qcom_enable_sntp_client (
	<i>A_BOOL enable</i>	Enable/disable SNTP
)	
Description	Enable/disable SNTP at run time.	
Return value	0 on success, -1 on failure.	

7.3.28 qcom_tcp_set_exp_backoff

Definition	Set the maximum number of TCP backff retries.	
Prototype	A_STATUS qcom_tcp_set_exp_backoff (
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1 (default 0)
	<i>A_INT32 retries,</i>	Number of retires (4 – 12)
)	
Return value	A_OK / A_ERROR	

7.3.29 qcom_ip4_route

Definition	Add, delete or query IPv4 route.	
Prototype	A_STATUS qcom_ip4_route (
	<i>A_UINT8 device_id,</i>	Unused (default 0)

	<i>A_UINT32 cmd,</i>	Cmd – ROUTE_ADD (0) ROUTE_DEL (1) ROUTE_SHOW (2)
	<i>A_UINT32* addr,</i>	Pointer to IPv4 Address
	<i>A_UINT32* subnet,</i>	Pointer to subnet mask
	<i>A_UINT32* gw,</i>	Pointer to Gateway IPv4 address
	<i>A_UINT32* ifindex,</i>	Pointer to interface index
	<i>IPV4_ROUTE_LIST_T* rtlist</i>)	List of routes, returned with ROUTE_SHOW option
Return value	A_OK / A_ERROR	

7.3.30 qcom_ip6_route

Definition	Add, delete or query IPv6 route.	
Prototype	<i>A_STATUS qcom_ip6_route (</i>	
	<i>A_UINT8 device_id,</i>	Unused (default 0)
	<i>A_UINT32 cmd,</i>	Cmd – ROUTE_ADD (0) ROUTE_DEL (1) ROUTE_SHOW (2)
	<i>A_UINT8* addr,</i>	Pointer to IPv6 Address
	<i>A_UINT32* prefixLen,</i>	Prefix Length
	<i>A_UINT8* gw,</i>	Pointer to Gateway IPv6 address
	<i>A_UINT32* ifindex,</i>	Pointer to interface index (used with ROUTE_DEL option)
	<i>IPV6_ROUTE_LIST_T* rtlist</i>)	List of routes, returned with ROUTE_SHOW option
Return value	A_OK / A_ERROR	

7.3.31 qcom_tcp_conn_timeout

Definition	Set the the TCP connection timeout value.	
Prototype	<i>A_STATUS qcom_tcp_conn_timeout (</i>	
	<i>A_INT32 timeout_val</i>	Connection timeout value in seconds
	<i>)</i>	
Return value	A_OK / A_ERROR	

7.3.32 qcom_socket()

Definition	Create a socket.	
Prototype	<i>int qcom_socket (</i>	
	<i>int family,</i>	Protocol family (AF_INET, AF_INET6)
	<i>int type,</i>	Socket type (SOCK_DGRAM or SOCK_STREAM)
	<i>int protocol</i>	Protocol (usually 0)
	<i>)</i>	

Description	This call is blocked until a socket handle is returned. If no response is received or socket creation fails, error code is returned.
Return value	Socket handle on success, -1 on failure

7.3.33 qcom_connect()

Definition	Connect to a socket.	
Prototype	int qcom_connect(
	int s,	Socket handle
	struct sockaddr* add,	<pre>struct sockaddr { u_short sa_family; /* address family */ #ifdef IP_V6 /* V6 only or dual stacks */ char sa_data[32]; /* big enough for unpacked sockaddr_in6 */ #else /* Ancient IPv4 only version */ char sa_data[14]; /* up to 14 bytes of direct address */ #endif };</pre>
	int addrlen	Length of sockaddr structure
)	
Description	<p>If the socket type is UDP, specify the peer to be associated with, the address to which datagrams are sent, and the address from which datagrams are received.</p> <p>If the socket type is TCP, this call attempts to connect to another socket.</p> <p>The call is blocked until a connection is established or error is returned.</p>	
Return value	0 on success, -1 on failure	

7.3.34 qcom_bind()

Definition	Assign a local socket address to an unnamed socket.	
Prototype	int qcom_bind(
	int s,	Socket handler
	struct sockaddr* addr,	Sockaddr structure indicating the local address to bind with. The address is provided in the name field in the form of a sockaddr (or sockaddr_6) structure.
	int addrlen	Length of sockaddr structure
)	
Return value	0 on success, -1 on failure	

7.3.35 qcom_listen()

Definition	Listen on incoming connections.	
Prototype	int qcom_listen(
	int s,	Socket handle
	int backlog	Maximum length of pending connections

)	
Description	If a connection request arrives, a <code>qcom_accept</code> call is triggered to accept it. <code>qcom_listen</code> is used only with socket type <code>SOCK_STREAM</code> . The backlog parameter defines the maximum length of the queue for pending connections. If a connection request arrives while the queue is full, the client receives error code <code>ECONNREFUSED</code> .	
Return value	0 on success, -1 on failure	

7.3.36 `qcom_accept()`

Definition	Accept incoming connections on an open socket.	
Prototype	<code>int qcom_accept(</code>	
	<code>int s,</code>	Socket handle
	<code>struct sockaddr* addr,</code>	Sockaddr structure indicating the local address to accept
	<code>int *addrlen</code>	Length on sockaddr structure
	<code>)</code>	
Description	After <code>qcom_listen()</code> returns success (socket created and bound to an address), this API extracts the first connection from the queue of pending connections. This API is used only with socket type <code>SOCK_STREAM</code> . The call is blocked if no connection is present or the socket is blocking.	
Return value	Non-negative socket descriptor on success, -1 on error	

7.3.37 `qcom_setsockopt()`

Definition	Set options for an existing socket.	
Prototype	<code>int qcom_setsockopt(</code>	
	<code>int s,</code>	Socket handle
	<code>int level,</code>	<code>SOL_SOCKET</code>
	<code>int name,</code>	<code>TCP_MAXSEG</code> <code>IP_ADD_MEMBERSHIP</code> <code>IP_DROP_MEMBERSHIP</code>
	<code>void* arg,</code>	Option value
	<code>int arglen</code>	Option length
	<code>)</code>	
Return value	0 on success, -1 on failure	

7.3.38 `qcom_getsockopt()`

Definition	Get options of an existing socket.	
Prototype	<code>int qcom_getsockopt(</code>	
	<code>int s,</code>	Socket handle
	<code>int level,</code>	<code>SOL_SOCKET</code>

	<i>int name,</i>	TCP_MAXSEG IP_ADD_MEMBERSHIP IP_DROP_MEMBERSHIP
	<i>void* arg,</i>	Option value
	<i>int arglen</i>	Option length
)	
Return value	0 on success, -1 on failure	

7.3.39 qcom_sendto()

Definition	Send data on a datagram socket.	
Prototype	int qcom_sendto(
	<i>int s,</i>	Socket handler
	<i>char* buf,</i>	A pointer to data
	<i>int len,</i>	Payload length
	<i>int flags,</i>	Send flags, usually 0
	<i>struct sockaddr* to,</i>	Sockaddr structure (v4 or v6)
	<i>int tolen</i>	Length of sockaddr
)	
Description	The destination address must be specified in the “to” parameter. This API works for both IPv4 and IPv6 with the sockaddr and sockaddr_6 structures respectively in the name parameter.	
Return value	On success: Number of bytes transmitted to the target. On failure: <ul style="list-style-type: none"> ▪ 100: No Tx buffers available. Caller is required to sleep and retry. ▪ 1: Fatal error, or packet length exceeded 1576 bytes. 	

7.3.40 qcom_send()

Definition	Send data on a stream socket.	
Prototype	int qcom_send(
	<i>int s,</i>	Socket handle
	<i>char* buf,</i>	A pointer to data
	<i>int len,</i>	Payload length
	<i>int flags,</i>	Send flags, usually 0
)	
Description	This API is used to send data on a stream socket in connected state. The API works for both IPv4 and IPv6.	
Return value	On success: Number of bytes transmitted to the target. On failure: <ul style="list-style-type: none"> ▪ -100: No Tx buffers available. Caller is required to sleep and retry. ▪ -1: Fatal error, or packet length exceeded 1576 bytes. ▪ Others: socket error number 	

7.3.41 qcom_recvfrom()

Definition	Receive data on a datagram socket.	
Prototype	int qcom_recvfrom(
	int s,	Socket handle
	char* buf,	A pointer to data
	int len,	Payload length
	int flags,	Send flags
	struct sockaddr* from,	Sockaddr structure (v4 or v6)
	int *fromlen	Length of sockaddr
)	
Description	<p>If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a sufficiently large buffer to receive the payload. If the buffer length is smaller than the available payload, the API performs a partial copy and drops the rest of the payload.</p>	
Return value	Number of bytes received on success, -1 on error, and 0 when socket is not found.	

7.3.42 qcom_recv()

Definition	Receive data on a stream socket.	
Prototype	int qcom_recv(
	int s,	Socket handle
	char* buf,	A pointer to data
	int len,	Payload length
	int flags,	Send flags
	}	
Description	<p>The socket must be in connected state for receive operation. If no packet is available, the socket is blocked until it is set to non-blocking. The application must provide a valid buffer to receive the payload. If the buffer length is smaller than the available payload, the API performs a partial copy and holds the rest of the payload for a subsequent call to the API.</p>	
Return value	Number of bytes received on success, -1 on error, and 0 when socket is closed by a peer.	

7.3.43 qcom_socket_close()

Definition	Close a socket.	
Prototype	int qcom_socket_close(
	int s	Socket handle
)	
Description	If the socket does not exist, error is returned.	
Return value	0 on success, -1 on error.	

7.3.44 qcom_select()

Definition	Allow an application thread to block on a given socket for a specified period.	
Prototype	int qcom_select(
	int max,	Socket handle + 1
	q_fd_set *read,	typedef struct { unsigned long fds_bits[1]; } q_fd_set;
	q_fd_set *write,	
	q_fd_set *ex,	
	struct timeval* timeout	struct timeval { __time_t tv_sec; /* Seconds. */ __suseconds_t tv_usec; /* Microseconds. */ };
)	
Description	Check for activities on the specified socket. The activities can be arrival of a packet at the receive queue. Supporting APIs for setting and clearing the fd are FD_ZERO, FD_CLR, FD_SET, FD_ISSET. For example, implementation, refer to “swat_udp_rx_data” function in SDK shell demo.	
Return value	Number of active sockets on success, 0 on no activity	

7.3.45 qcom_http_server()

Definition	Start/Stop HTTP/HTTPS server	
Prototype	int qcom_http_server(
	A_INT32 enable,	Enable: 0 = Stop HTTP server 1 = Start HTTP server 2 = Stop HTTPS server 3 = Start HTTPS server
	void *ssl_ctx	For HTTP with SSL case. It is NULL for normal HTTP without SSL.
)	
Return value	0 on success, -1 on error.	

7.3.46 qcom_http_server_method()

Prototype	A_INT32 qcom_http_server_method(A_INT32 <i>command</i> ,	
	A_UINT8 * <i>pagename</i> ,	0 = POST; 1 = GET
	A_UINT8 * <i>objname</i> ,	Page name from/to which the data is to get/post
	A_INT32 <i>objtype</i> ,	Name of the object in HTML file
	A_INT32 <i>objlen</i> ,	Type of the object (string/integer/Boolean)
	A_UINT8 * <i>value</i>	Object length 1 = Boolean; 4 = Integer; String length = string
)	Value of the object
Description	Get/post the values from/to the HTTP server. For GET operation, make sure sufficient memory is allocated before calling this API.	
Return value	0 on success, -1 on failure.	

7.3.47 qcom_http_set_post_cb

Definition	Set callback for post events.	
Prototype	A_STATUS qcom_http_set_post_cb(void * <i>cxt</i> ,	HTTP connection context
	void * <i>callback</i>	The callback function
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.48 qcom_http_set_get_cb

Definition	Set callback for get events.	
Prototype	A_STATUS qcom_http_set_get_cb(void * <i>cxt</i> ,	HTTP connection context
	void * <i>callback</i>	The callback function
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.49 qcom_http_set_custom_uri

Definition	Set callback for get events.	
Prototype	A_STATUS qcom_http_set_custom_uri(A_UINT8 <i>device_id</i> ,	Device ID
	const char* <i>new_uri</i>	The Customer URI string array pointer
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.50 qcom_http_get_datasend

Definition	Set callback for get events.	
Prototype	A_STATUS qcom_http_get_datasend(A_UINT16 sess_index, A_UINT16 data_len, A_UINT8 *url_str, A_UINT8 *data, A_UINT16 header_len, A_UINT8 *header_data)	
		HTTP session index from "GET" event
		The customer allocated HTTP body buffer len
		The HTTP session URL string
		The customer allocated HTTP body buffer
		The customer allocated HTTP header buffer len
		The customer allocated HTTP header buffer
Return value	A_OK on success, A_ERROR on failure	

7.3.51 qcom_http_redirect_unknown_url_enable

Definition	Configure HTTP server redirecting unknown URL function	
Prototype	A_STATUS qcom_http_redirect_unknown_url_enable(A_UINT8 device_id, A_UINT8 enable,)	
		Current device ID
		Enable/disable redirecting function
Return value	A_OK on success, A_ERROR on failure	

7.3.52 qcom_http_server_add_redirected_page

Definition	Add redirected page	
Prototype	A_STATUS qcom_http_server_add_redirected_page(A_UINT16 header_len, A_UINT8 *header A_UINT16 data_len A_UINT8 *data Char *url ,)	
		Header length of redirected HTML
		Header of redirected HTML
		Data length of redirected HTML
		Data of redirected HTML
		URL of HTML, no more than 256
Return value	A_OK on success, A_ERROR on failure	

7.3.53 qcom_restrict_http_request

Definition	Enable/disable http request restriction function	
Prototype	A_STATUS qcom_restrict_http_request (A_UINT8 device_id, A_UINT8 enable	
		Device ID
		Enable or disable

)	
Return value	A_OK on success, A_ERROR on failure	

7.3.54 qcom_http_set_redirected_url

Definition	Add url name for redirecting html	
Prototype	A_STATUS qcom_http_set_redirected_url (
	A_UINT8 device_id,	Device ID
	Char * url	Newly added URL for redirecting HTML
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.55 qcom_http_client_method

Definition	Configure HTTP client	
Prototype	A_STATUS qcom_http_client_method(
	A_UINT32 cmd,	0 = HTTPC_CONNECT_CMD 1 = HTTPC_GET_CMD 2 = HTTPC_POST_CMD 3 = HTTPC_DATA_CMD 4 = HTTPC_DISCONNECT_CMD 5 = HTTPC_CONNECT_SSL_CMD 6 = HTTPC_HEADER_CMD 7 = HTTPC_CLEAR_HEADER_CMD 8 = HTTPC_BODY_CMD 9 = HTTPC_PUT_CMD 10 = HTTPC_PATCH_CMD
	A_UINT8 *url,	The content of this pointer is the URL of HTTP server or its IP address. Normally its length is less than 256 bytes.
	A_UINT8 *data,	If the function is called with connect command, the data would be HTTP connect port. If it is called with HEADER_CMD or DATA_CMD, it would be data from user input.
	A_UINT8 *value,	Output of the HTTP client request
	void *ssl_ctx	For HTTP with SSL, the SSL context with data instead of NULL is passing to differentiate between HTTPS and HTTP in the lower layers. For normal HTTP, the SSL context is NULL.
)	
Return value	A_OK on success, A_ERROR on failure	
Comments	Deprecated API. Use new HTTP client APIs for nonlegacy system (see sections 7.3.58 through 7.3.64).	

7.3.56 qcom_http_client_body

Definition	Configure HTTP client	
Prototype	A_STATUS qcom_http_client_body(
	A_UINT32 cmd,	8 = HTTPC_BODY_CMD
	A_UINT8 *body,	HTTP client body content
	A_UINT32 bodylen	HTTP client body len
)	
Return value	A_OK on success, A_ERROR on failure	
Comments	Deprecated API. Use new http client APIs for nonlegacy system (see sections 7.3.58 through 7.3.64).	

7.3.57 qcom_http_client_register_cb

Definition	Register or de-register HTTP client callback	
Prototype	A_STATUS qcom_http_client_register_cb(
	http_client_cb_t callback	Callback function. Null indicates callback deregistration.
	void* arg	User parameter used in callback function
)	
Return value	A_OK on success, A_ERROR on failure	
Comments	Deprecated API. Use new HTTP client APIs for nonlegacy system (see sections 7.3.58 through 7.3.64).	

7.3.58 qcom_http_client_connect

Definition	Create a HTTP client and connect to server	
Prototype	A_INT32 qcom_http_client_connect(
	const A_CHAR *server,	HTTP server address
	A_UINT16 port,	HTTP server port
	A_UINT32 timeout	Timeout of session method, unit in milliseconds. 0 means wait forever in callback mode. In callback disable mode, the user thread will be blocked 60 s at most.
	void* ssl_ctx	Pointer of SSL context
	http_client_cb_t callback	Callback function of this client
	typedef void (*http_client_cb_t)(void* arg,	User arg

		<i>A_INT32 state,</i>	-6 = HTTPC_RX_ERROR_RX_HTTP_HEADER -5 = HTTPC_RX_ERROR_RX_HTTP_HEADER -4 = HTTPC_RX_ERROR_TIMEOUT -3 = HTTPC_RX_ERROR_NO_BUFFER -2 = HTTPC_RX_CONNECTION_CLOSED -1 = HTTPC_RX_ERROR_CONNECTION_CLOSED 0 = HTTPC_RX_FINISHED 1 = HTTPC_RX_MORE_DATA
		<i>void* value</i>	HTTP response data if state = 0 or 1
)	
		Description	User callback function definition type
		<i>Void* arg</i>	Arg of the callback function
)		
	Return value	Handler of created HTTP client on success, A_ERROR on error. NOTE: If exceeding the maximum number of session, return A_NO_HTTP_SESSION.	

7.3.59 qcom_http_client_disconnect

Definition	HTTP client disconnect from server	
Prototype	A_STATUS qcom_http_client_disconnect(
	<i>A_INT32 client</i>	The client to disconnect
)	
Return value	A_OK on success, A_ERROR on error	

7.3.60 qcom_http_client_request

Definition	HTTP client execute command and waiting for result	
Prototype	A_STATUS qcom_http_client_request(
	<i>A_INT32 client,</i>	The client to execute command
	<i>A_UINT32 cmd,</i>	Cmd to execute 20 = HTTP_CLIENT_GET_CMD 21 = HTTP_CLIENT_POST_CMD 22 = HTTP_CLIENT_PUT_CMD 23 = HTTP_CLIENT_PATCH_CMD
	<i>const A_CHAR *url,</i>	URL to access
	<i>A_CHAR *output</i>	Pointer from user to receive data from server, optional parameter.
)	
Return value	A_OK on success, A_ERROR on error	

7.3.61 qcom_http_client_set_body

Definition	HTTP client set body	
Prototype	A_STATUS qcom_http_client_set_body(

	<i>A_INT32 client,</i>	Client to set body
	<i>const A_CHAR *body,</i>	Pointer of the body to be set
	<i>A_UINT32 bodylen</i>	Length of the body to be set
)	
Return value	A_OK on success, A_ERROR on error	

7.3.62 qcom_http_client_set_param

Definition	HTTP client set parameter	
Prototype	A_STATUS qcom_http_client_set_param(<i>A_INT32 client,</i>	Client to set parameter
	<i>const A_CHAR *key,</i>	Pointer of key to be set
	<i>const A_CHAR *value</i>	Pointer of value to be set
)	
Return value	A_OK on success, A_ERROR on error	

7.3.63 qcom_http_client_add_header

Definition	HTTP client add header	
Prototype	A_STATUS qcom_http_client_add_header(<i>A_INT32 client,</i>	Client to add header
	<i>const A_CHAR *header,</i>	Pointer of header to be added
	<i>const A_CHAR *value</i>	Pointer of value to be added to head
)	
Return value	A_OK on success, A_ERROR on error	

7.3.64 qcom_http_client_clear_header

Definition	HTTP client clear header	
Prototype	A_STATUS qcom_http_client_clear_header(<i>A_INT32 client</i>	Client to clear header
)	
)	
Return value	A_OK on success, A_ERROR on error	

7.3.65 qcom_ota_upgrade()

Definition	This API allows user to either upgrade target (QCA4010/QCA4012) or host from an external server.	
Prototype	A_INT32 qcom_ota_upgrade(<i>A_UINT8 device_id,</i>	Device ID
	<i>A_UINT32 serverip,</i>	Server IP address
	<i>A_CHAR* filename,</i>	Filename
)	

	<i>A_UINT8 mode,</i>	0 = Firmware OTA download 1 = Host OTA download
	<i>A_UINT8 preserve_last,</i>	0 = OTA can overwrite the current partition. 1 = OTA is not allowed to overwrite the current partition. The preserve_last parameter is mainly used when only one alternative partition is available (which happens to be the current one). Unsuccessfully updating that partition results in falling back to the default partition which can be old.
	<i>A_UINT8 protocol,</i>	0 = TFTP
	<i>A_UINT32* resp_code,</i>	OTA Service updates resp_code and size of the downloaded OTA image.
	<i>A_UINT32 *length</i>	
)	
Description	Initialize OTA Service.	
Return value	A_OK on success, A_ERROR on error	

7.3.66 qcom_ota_done()

Prototype	A_INT32 qcom_ota_done()	
Description	Terminate or close the OTA Service.	
Return value	A_OK on success, refer to section 3.10.1.4 for error codes.	

7.3.67 qcom_ota_session_start

Definition	Initialize OTA upgrade session.	
Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_session_start(

	<i>A_UINT32 flags,</i>	<p>QCOM_OTA_TARGET_FIRMWARE_UPGRADE</p> <ul style="list-style-type: none"> Set: The application can upgrade a target firmware partition. The OTA engine selects the upgrade partition and activates the image (rank and magic) once the session completes. Not set: The application can upgrade a partition of some other binary blob. The OTA upgrade engine does not know anything about the image and does not attempt to modify it. <p>QCOM_OTA_PRESERVE_LAST</p> <p>This flag is valid for target firmware image upgrade only. It is not applicable for two-partition system.</p> <ul style="list-style-type: none"> Set: The OTA engine does not overwrite the last active target firmware partition. In a two-partition system, it fails after the first successful upgrade because the last active partition is always the second partition (golden partition is never updated). <p>QCOM_OTA_ERASING_RW_DSET</p> <ul style="list-style-type: none"> Set: Erase all RW data sets in the selected partition if it is also the active partition. Default behavior is to retain RW data sets.
	<i>A_UINT32 partition_index</i>	<p>Index of the requested partition to be upgraded, or OTA_PARTITION_AUTO to allow the OTA engine to automatically select the partition.</p> <ul style="list-style-type: none"> Request to upgrade partition 0 always fails. If the image type is not target firmware, a partition index must be specified. OTA_PARTITION_AUTO cannot be used in this case. If the image type is target firmware, the OTA engine does not allow specifying an index (ignored), and OTA_PARTITION_AUTO must be used for this case. The OTA engine will then select an empty partition (first 4 bytes equal to 0xFFFF FFFF). If not found, the OTA engine selects the target firmware partition with the lowest rank. If the image type is not firmware and the requested partition index contains active target firmware, the operation will fail.
)	
Return value	<p>QCOM_OTA_OK on success; A specific error code in QCOM_OTA_STATUS_CODE_t on failure.</p>	

7.3.68 qcom_ota_partition_get_size

Definition	Allow the application to get the partition size. This is required by the plug-in to avoid overflowing. The API is blocking and returns 0 if no active OTA session is present.	
Prototype	A_UINT32	
	qcom_ota_partition_get_size()	
Return value	Size of the partition in bytes	

7.3.69 qcom_ota_partition_erase

Definition	Allow the application to erase a partition. The partition is selected by the ota_session_start() API. The API is blocking and returns once the partition is erased. The RW data sets in the selected partition is preserved if the QCOM_OTA_ERASING_RW_DSET flag is not set in ota_session_start().	
Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_partition_erase()	
)	
Return value	QCOM_OTA_OK: Partition erased successfully QCOM_OTA_ERR_NO_ACTIVE_OTA_SESSION: OTA session not started QCOM_OTA_ERR_FLASH_ERASE_ERROR: Invalid partition	

7.3.70 qcom_ota_partition_erase_sectors

Definition	Allow the application to erase a partition. The partition is selected by the ota_session_start() API. The API is blocking and returns once the partition is erased to offset_next. The RW data sets in the selected partition is preserved if the QCOM_OTA_ERASING_RW_DSET flag is not set in ota_session_start().	
Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_partition_erase_sectors(<i>A_UINT32 offset_next</i>)	
		Offset from the start of the partition. This will erase partition from erased offset (last time) to offset_next based on sector size boundary. This can be called before call qcom_ota_partition_write_data.
)	
Return value	QCOM_OTA_OK: Partition erased successfully QCOM_OTA_ERR_NO_ACTIVE_OTA_SESSION: OTA session not started QCOM_OTA_ERR_FLASH_ERASE_ERROR: Invalid partition	

7.3.71 qcom_ota_parse_image_hdr

Definition	Allow the application to parse a standard QCOM OTA upgrade header. This API cannot be used to parse any other headers. If the header is parsed correctly, the offset to the first byte of the image is returned.	
Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_parse_image_hdr(<i>A_UINT8 *header,</i> <i>A_UINT32 *offset</i>)	
		Pointer to QCOM OTA header
		Output: offset of the first byte of the image data
)	
Return value	QCOM_OTA_ERR_IMAGE_HDR_INCORRECT: Invalid header format QCOM_OTA_ERR_IMAGE_OVERFLOW: Image overflowing the partition QCOM_OTA_OK: Header parsed correctly	

7.3.72 qcom_ota_partition_verify_checksum

Definition	Allow the application to verify the checksum of the image. The <code>qcom_ota_parse_image_hdr()</code> API is required to parse the header. This API does not work if a different header is used.	
Prototype	<code>QCOM_OTA_STATUS_CODE_t</code> <code>qcom_ota_partition_verify_checksum(</code>	
	<code>)</code>	
Return value	<code>QCOM_OTA_ERR_IMAGE_CHECKSUM_INCORRECT</code> : Checksum incorrect <code>QCOM_OTA_OK</code> : Image checksum correct	

7.3.73 qcom_ota_partition_read_data

Definition	Allow the application to read data from the partition. This call will fail if no OTA session is active. The partition is selected by the <code>ota_session_start()</code> API. This API does not allow reading data outside of the partition bounds. This API is blocking and returns once the flash read operation is complete.	
Prototype	<code>A_INT32</code> <code>qcom_ota_partition_read_data(</code>	
	<code>A_UINT32 offset</code>	Offset from the start of the partition
	<code>A_UINT8 *buf</code>	Point to buffer
	<code>A_UINT size</code>	Buffer size/amount of bytes to read
	<code>)</code>	
Return value	Amount of bytes to be read and written to the buffer, or -1 on failure	

7.3.74 qcom_ota_partition_write_data

Definition	Allow the application to write data to the partition. This call will fail if no OTA session is active. The partition is selected by the <code>ota_session_start()</code> API. This API does not allow writing data outside of the partition bounds. This API is blocking and returns once the flash write operation is complete.	
Prototype	<code>QCOM_OTA_STATUS_CODE_t</code> <code>qcom_ota_partition_write_data(</code>	
	<code>A_UINT32 offset</code>	Offset from the start of the partition
	<code>A_UINT8 *buf</code>	Point to buffer
	<code>A_UINT32 size</code>	Buffer size/amount of bytes to read
	<code>A_UINT32 *ret_size</code>	Amount of bytes written to the flash or -1 on failure
	<code>)</code>	
Return value	<code>QCOM_OTA_OK</code> : Flash write successful <code>QCOM_OTA_ERR_NO_ACTIVE_OTA_SESSION</code> : OTA session not started <code>QCOM_OTA_ERR_IMAGE_OVERFLOW</code> : Offset out of the bounds of the partition	

7.3.75 qcom_ota_session_end

Definition	Allow the application to end an OTA upgrade session. The caller specifies the result (OK or an error code). The OTA engine frees all resources and terminates the session. If the upgrade was successful (indicated by the parameter) and the image was a target firmware image
-------------------	---

Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_session_end(
	<i>A_UINT32 good_image</i>	Indicates whether the image has passed the caller's validations (the image marked valid)
)	
Return value	QCOM_OTA_ERR_NO_ACTIVE_OTA_SESSION: OTA session not started QCOM_OTA_COMPLETED: Success. The image was updated.	

7.3.76 qcom_ota_partition_format

Definition	Allow the application to format a specific partition. This API is used to revive a corrupted partition. Partition 0 (golden partition) and the currently active firmware partition cannot be formatted.	
Prototype	QCOM_OTA_STATUS_CODE_t qcom_ota_partition_format(
	<i>A_UINT32 partition_index</i>	Partition to be formatted
)	
Return value	QCOM_OTA_OK: Partition formatted successfully QCOM_OTA_ERR_INVALID_PARTITION_INDEX: Index 0 (Golden image) provided QCOM_OTA_ERR_FLASH_ERASE_ERROR: Partition formatting failed	

7.3.77 qcom_read_ota_area

Definition	Read data (in bytes) from the offset in OTA area.	
Prototype	A_STATUS qcom_read_ota_area(
	<i>A_UINT32 offset,</i>	Offset to the OTA area
	<i>A_UINT32 size,</i>	Data size in bytes
	<i>A_UCHAR *buffer,</i>	The buffer to store the read content
	<i>A_UINT32 *ret_len</i>	The return data length
)	
Return value	A_OK on success, A_ERROR on failure	

7.3.78 qcom_set_ping_id

Definition	Set Ping ID.	
Prototype	A_STATUS qcom_set_ping_id (
	<i>A_UINT32 ping_id</i>	Ping ID to be set
)	
Return value	A_OK on success, A_ERROR on error	

7.3.79 qcom_dhcpc_set_timeout

Definition	Set DHCP Client timeout value.	
Prototype	A_STATUS qcom_dhcpc_set_timeout(

	<i>A_UINT8 device_id</i>	Device ID to be set
	<i>A_UINT32 timeout</i>	The timeout value to be set
)	
Description	Timeout Value while waiting for a DHCP OFFER/ACK/NAK. If timeout happens, retransmit the packet.	
Return value	A_OK on success, A_ERROR on error	

7.3.80 qcom_dhcpc_register_cb

Definition	Register DHCP Client callback.	
Prototype	A_STATUS qcom_dhcpc_register_cb (
	<i>A_UINT8 device_id</i>	Device ID
	<i>void *callback</i>	Callback function. See details as below.
)	
	<i>void (*callback) (</i>	
	<i>A_UINT32 ip,</i>	IPv4 address
	<i>A_UINT32 mask,</i>	IPv4 mask
	<i>A_UINT32 gw</i>	Gateway address
)	
Return value	0 on success, -1 on failure.	

7.3.81 qcom_dhcps_register_cb

Definition	Register DHCP Server callback.	
Prototype	A_STATUS qcom_dhcps_register_cb (
	<i>A_UINT8 device_id</i>	Device ID
	<i>void *callback</i>	Callback function. The details is below.
)	
	<i>void (*callback) (</i>	
	<i>A_UINT8 *mac,</i>	MAC address
	<i>A_UINT32 ip</i>	IPv4 address
)	
Return value	0 on success, -1 on failure.	

7.4 Wireless Networking APIs

7.4.1 qcom_get_scan

Definition	Get the scan results.	
Prototype	A_STATUS qcom_get_scan (

	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>QCOM_BSS_SCAN_INFO** buf,</i>	Pointer to buffer that contains scan results
	<i>A_INT16* numResults</i>	Pointer to number of results
)	
Description	This API will block until scan is complete. User must not allocate or free “buf” parameter, it will point to the library’s scan buffer.	
Return value	A_OK / A_ERROR	

7.4.2 qcom_set_ssid

Definition	Set SSID.	
Prototype	A_STATUS qcom_get_ssid (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_CHAR *pssid</i>	A pointer to SSID (length < 32)
)	
Description	Set the SSID. This API must be called before calling qcom_set_scan or qcom_commit.	
Return value	A_OK / A_ERROR	

7.4.3 qcom_get_ssid

Definition	Get SSID.	
Prototype	A_STATUS qcom_get_ssid (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_CHAR *pssid</i>	A pointer to SSID (length < 32)
)	
Return value	A_OK / A_ERROR	

7.4.4 qcom_set_connect_callback

Definition	Set the call back function which will be invoked on connection state change.	
Prototype	A_STATUS qcom_set_connect_callback (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>void *callback</i>	Call back function. The prototype of the function is: void fn(A_UINT8 device_id, A_INT32 value) value = 1 (Successful connection) = 0 (Disconnect)
)	
Return value	A_OK / A_ERROR	

7.4.5 qcom_sta_connect_legacy

Definition	Scan first and then connect to an AP using the parameters from the scan.	
Prototype	A_STATUS qcom_sta_connect_legacy (
	A_UINT8 device_id,	Interface ID (default 0)
	A_CHAR ssid[]	Pointer to SSID
)	
Return value	A_OK / A_ERROR	

7.4.6 qcom_commit

Definition	Connect to an AP or start soft AP mode with the security configuration by user.	
Prototype	A_STATUS qcom_commit (
	A_UINT8 device_id,	Interface ID (default 0)
)	
Description	<p>This API initiates the connect procedure. It can also be used to start a soft AP. The user must do the following before calling commit:</p> <ol style="list-style-type: none"> 1. Set Callback function. 2. Set Mode (Station/ soft AP/Ad-Hoc). 3. Set SSID. 4. Set security parameters (optional). 	
Return value	A_OK / A_ERROR	

7.4.7 qcom_sta_reconnect_start

Definition	Start reconnect action at an interval specified by user.	
Prototype	A_STATUS qcom_sta_reconnect_start (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 seconds	Reconnect interval in seconds
)	
Return value	A_OK / A_ERROR	

7.4.8 qcom_set_keep_alive

Definition	Enable/disable keep-alive function and set keep-alive interval (Station mode only).	
Prototype	A_STATUS qcom_set_keep_alive (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 keep_alive_interval	Keep-alive interval in seconds. 0 = Disable keep-alive. Nonzero = Enable keep-alive. The setting takes effect from the next connection with an AP.
)	
Return value	A_OK / A_ERROR	

7.4.9 qcom_roam_mode_enable

Definition	Enable/disable roaming function.	
Prototype	A_STATUS qcom_roam_mode_enable (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT8 enable,	1 = Enable roaming; 0 = Disable roaming.
)	
Return value	A_OK / A_ERROR	

7.4.10 qcom_sta_reconnect_stop

Definition	Stop reconnect action.	
Prototype	A_STATUS qcom_sta_reconnect_stop (
	A_UINT8 device_id,	Interface ID (default 0)
)	
Return value	A_OK / A_ERROR	

7.4.11 qcom_sta_get_rssi

Definition	Get RSSI value for link quality (SNR).	
Prototype	A_STATUS qcom_sta_get_rssi (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT8 *prssi	Link quality (SNR in dB)
)	
Return value	A_OK / A_ERROR	

7.4.12 qcom_sta_set_listen_time

Definition	Set listen interval.	
Prototype	A_STATUS qcom_sta_set_listen_time (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 time	Time interval in TUs for the device to wake up and listen for traffic. 1 TU = 1024 μ s
)	
Return value	A_OK / A_ERROR	

7.4.13 qcom_sta_get_listen_time

Definition	Get listen interval.	
Prototype	A_STATUS qcom_sta_get_listen_time (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *ptime	A pointer to of listen time interval
)	

Return value	A_OK / A_ERROR
---------------------	----------------

7.4.14 qcom_ap_set_beacon_interval

Definition	Set Beacon interval (Soft AP mode).	
Prototype	A_STATUS qcom_ap_set_beacon_interval (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 beacon_interval	Beacon interval in TUs (1 TU = 1024 μ s)
)	
Description	The default beacon interval is 100 TUs (102.4 ms).	
Return value	A_OK / A_ERROR	

7.4.15 qcom_ap_get_beacon_interval

Definition	Get beacon interval (SoftAP mode).	
Prototype	A_STATUS qcom_ap_get_beacon_interval(
	A_UINT8 device_id,	Interface ID
	A_UINT16 *pbeacon_interval	A pointer to beacon interval
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.16 qcom_ap_set_inact_time

Definition	Set inactive time (SoftAP mode).	
Prototype	A_STATUS qcom_ap_set_inact_time(
	A_UINT8 device_id,	Interface ID
	A_UINT16 minutes	Inactive time in minutes
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.17 qcom_ap_get_sta_info

Definition	Get information of the stations that are connected to the SoftAP.	
Prototype	A_STATUS qcom_ap_get_sta_info(
	A_UINT8 device_id,	Interface ID
	A_UINT8 *pnnum,	A pointer to the number of the stations which are connected to the AP
	A_UINT8 *pmac	A pointer to the MAC of the stations which are connected to the AP
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.18 qcom_buffer_info_get

Definition	Get the buffer statistics for system debug.	
Prototype	A_STATUS qcom_buffer_info_get(
	A_UINT8 *pmac_cnt,	Rx free buffer count
	A_UINT8 *pmac_reap,	Rx used buffer count
	A_UINT8 *phtc_cnt,	Tx free buffer count
	A_UINT8 *phtc_reap,	Tx used buffer count
	A_UINT8 *pfw_cnt,	Internal free buffer count
	A_UINT8 *pfw_reap,	Internal used buffer count
	A_UINT8 *pfree_cnt,	Buffer left in the buffer pool
	A_UINT32 *phtc_get,	Tx buffer allocation statistics
	A_UINT32 *phtc_put,	Unused
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.19 qcom_firmware_version_get

Definition	Get the firmware version.	
Prototype	A_STATUS qcom_firmware_get(
	A_UINT32 *psize	A pointer to the free heap size
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.20 qcom_sec_set_wep_mode

Definition	Set WEP mode.	
Prototype	A_STATUS qcom_sec_set_wep_mode(
	A_UINT8 device_id,	Interface ID
	A_UINT32 mode	WEP mode 0 = Open; 1 = Shared
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.21 qcom_sec_get_wep_mode

Definition	Get WEP mode.	
Prototype	A_STATUS qcom_sec_get_wep_mode(
	A_UINT8 device_id,	Interface ID
	A_UINT32 *mode	A pointer to WEP mode 0 = Open; 1 = Shared
)	

Return value	A_OK on success, A_ERROR on failure
---------------------	-------------------------------------

7.4.22 qcom_thread_msleep

Definition	Thread sleep	
Prototype	void qcom_thread_msleep(
	<i>unsigned long ms</i>	Sleep time
)	
Return value		

7.4.23 qcom_set_lpl_enable

Definition	Enable or disable LPL function.	
Prototype	A_STATUS qcom_set_lpl_enable(
	<i>A_UINT8 device_id,</i>	Interface ID
	<i>A_UINT8 enable</i>	0 = Disable; 1 = Enable
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.24 qcom_set_gtx_enable

Definition	Enable or disable Green Tx function.	
Prototype	A_STATUS qcom_set_gtx_enable(
	<i>A_UINT8 device_id,</i>	Interface ID
	<i>A_UINT8 enable</i>	0 = Disable; 1 = Enable
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.25 qcom_set_rate

Definition	Set data rate.	
Prototype	A_STATUS qcom_set_rate(
	<i>A_UINT8 device_id,</i>	Interface ID
	<i>A_UINT8 mcs,</i>	0 = Disable; 1 = Enable
	<i>A_UINT8 rate</i>	If mcs is set to 0, rate can be set to 1, 2, 5, 11, 6, 9, 12, 18, 24, 36, 48, 54 Mbps. If mcs is set to 1, rate can be set to 0 to 7.
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.26 qcom_set_scan_timeout

Definition	Set scan timeout value.
-------------------	-------------------------

Prototype	Void qcom_set_scan_timeout (
	A_UINT8 sec	Timeout value
)	
Return value	None	

7.4.27 qcom_set_scan

Definition	Start a long or short channel scan.	
Prototype	A_STATUS qcom_set_scan(
	A_UINT8 device_id,	Interface ID
	qcom_start_scan_params_t *pScanParams	typedef struct _qcom_start_scan_params { A_BOOL forceFgScan; // Forces foreground scan A_UINT32 homeDwellTimeInMs; // Maximum duration in the home channel (milliseconds) A_UINT32 forceScanIntervalInMs; // Time interval between scans (milliseconds) A_UINT8 scanType; // 0 = full scan, 1 = short scan A_UINT8 numChannels; // Number of channels A_UINT16 channelList[QCOM_START_SCAN_PARAMS_CHANNEL_LIST_MAX]; // Number of channels } qcom_start_scan_params_t;
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.28 qcom_get_bss_entry_by_ssid

Definition	Get detailed information by the SSID from scan result.	
Prototype	QCOM_BSS_SCAN_INFO *qcom_get_bss_entry_by_ssid(
	A_UINT8 device_id,	Interface ID
	char ssid[]	SSID
)	

Return value	<p>Information of SSID. Return NULL if no matched SSID.</p> <pre>typedef struct _qcom_scan_info { unsigned char channel; //channel number. unsigned char ssid_len; //the length of ssid. unsigned char rssi; //received signal strength indicator. unsigned char security_enabled; //security enable/disable. unsigned short beacon_period; //beacon interval. unsigned char preamble; //short/long preamble. unsigned char bss_type; //ESS/IBSS type network. unsigned char bssid[6]; //bssid information. unsigned char ssid[32]; //ssid information. unsigned char rsn_cipher; //TKIP/CCMP/WEP/NA unsigned char rsn_auth; //PSK/1X/NA unsigned char wpa_cipher; //TKIP/CCMP/WEP/NA unsigned char wpa_auth; //PSK/1X/NA unsigned short caps; //capability info unsigned char wep_support; //not used unsigned char reserved; //keeps struct on 4-byte boundary } QCOM_BSS_SCAN_INFO, * QCOM_BSS_SCAN_INFO_PTR;</pre>
---------------------	--

7.4.29 qcom_uart_init

Definition	Initiate UART.	
Prototype	A_UINT32 qcom_uart_init(
)	
Return value	Number of UART.	

7.4.30 qcom_single_uart_init

Definition	Initiate a selected UART.	
Prototype	A_INT32 qcom_uart_init(
	A_CHAR *name,	UART port name
)	
Return value	0 on success, other values on failure.	

7.4.31 qcom_uart_set_buffer_size

Definition	Configure UART Tx/Rx buffer size before UART initialization.	
Prototype	A_INT32 qcom_uart_set_buffer_size (
	A_CHAR *name,	UART port name
	A_UINT32 tx_size	Set value of Tx buffer size
	A_UINT32 rx_size	Set value of Rx buffer size
)	
Return value	0 on success, other values on failure.	

7.4.32 qcom_get_uart_config

Definition	Get parameters of UART.	
Prototype	A_STATUS qcom_get_uart_config (
	A_CHAR *name,	UART port name
	qcom_uart_para *uartpara	typedef struct _uart_para { int BaudRate; // baud rate char number; // number of data bits char parity; // parity (0 = no parity; 1 = odd parity check; 2 = even parity check) default 0 char StopBits; //number of stop bits char FlowControl; //1 = support flow control } qcom_uart_para;
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.33 qcom_set_uart_config

Definition	Set parameters of UART.	
Prototype	A_STATUS qcom_set_uart_config (
	A_CHAR *name,	UART port name
	qcom_uart_para *uartpara	typedef struct _uart_para { int BaudRate; // baud rate char number; // number of data bits char parity; // parity (0 = no parity; 1 = odd parity check; 2 = even parity check) default 0 char StopBits; //number of stop bits char FlowControl; //1 = support flow control } qcom_uart_para;
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.34 qcom_uart_open

Definition	Open UART and get a file descriptor.	
Prototype	A_INT32 qcom_uart_open (
	A_CHAR *name	UART port name
)	
Return value	File descriptor	

7.4.35 qcom_uart_close

Definition	Close UART	
Prototype	A_INT32 qcom_uart_close (

	<i>A_INT32 fd</i>	File descriptor returned from <code>uart_open</code>
)	
Return value	0 on success, other values on failure	

7.4.36 `qcom_uart_read`

Definition	Read data from UART.	
Prototype	<i>A_INT32 qcom_uart_read (</i>	
	<i>A_INT32 fd,</i>	File descriptor returned from <code>uart_open</code>
	<i>A_CHAR *buff,</i>	Address of the buffer. The buffer size must be big enough for the data.
	<i>A_UINT32 *len</i>	Input length of the buffer. Output length of bytes that have been read.
)	
Return value	Bytes remain in the Rx buffer.	

7.4.37 `qcom_uart_write`

Definition	Write data to UART.	
Prototype	<i>A_INT32 qcom_uart_write (</i>	
	<i>A_INT32 fd,</i>	File descriptor returned from <code>uart_open</code>
	<i>A_CHAR *buff,</i>	Address of the buffer. The buffer size must be big enough for the data.
	<i>A_UINT32 *len</i>	Input length of the buffer. Output length of bytes that have been read.
)	
Return value	Bytes remain in the Tx buffer.	

7.4.38 `qcom_uart_wakeup_config`

Definition	Uart wakeup function configuration	
Prototype	<code>void qcom_uart_wakeup_config(</code>	
	<i>A_CHAR *name,</i>	UART port name
	<i>A_UINT8 mode,</i>	Wakeup method: break signal or UART flow control
	<i>A_UINT8 enable</i>	Enable or disable UART wakeup function
)	
Return value		

7.4.39 `qcom_mem_alloc`

Definition	Allocate a buffer	
Prototype	<code>void *qcom_mem_alloc(</code>	
	<i>unsigned int size,</i>	Size of the buffer to be allocated

)	
Return value	Pointer to the buffer on success, NULL on failure	

7.4.40 qcom_mem_free

Definition	Allocate a buffer	
Prototype	void *qcom_mem_free(
	void *p	Pointer to the buffer to be freed
)	
Return value		

7.4.41 qcom_ap_hidden_mode_enable

Definition	Enable/Disable SSID hidden mode (Soft AP mode).	
Prototype	A_STATUS qcom_ap_hidden_mode_enable (
	A_UINT8 device_id,	Interface ID (default 0)
	A_BOOL enable	1 = Enable; 0 = Disable
)	
Return value	A_OK / A_ERROR	

7.4.42 qcom_ap_set_max_station_number

Definition	Set the maximum number of stations allowed to connect.	
Prototype	A_STATUS qcom_ap_set_max_station_number (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 sta_num	Maximum number of stations
)	
Return value	A_OK / A_ERROR	

7.4.43 qcom_ap_set_flag

Definition	Set connect control flags.	
Prototype	A_STATUS qcom_ap_set_flag (
	A_UINT8 device_id,	Interface ID (default 0)

	<i>A_UINT32 flg</i>	<i>Control flag</i> CONNECT_ASSOC_POLICY_USER=0x0001, CONNECT_SEND_REASSOC =0x0002, CONNECT_IGNORE_WPAx_GROUP_CIPHER = 0x0004, CONNECT_PROFILE_MATCH_DONE=0x0008, CONNECT_IGNORE_AAC_BEACON=0x0010, CONNECT_CSA_FOLLOW_BSS=0x0020, CONNECT_DO_WPA_OFFLOAD=0x0040, CONNECT_DO_NOT_DEAUTH=0x0080, CONNECT_WPS_FLAG=0x0100, CONNECT_IGNORE_BSSID_HINT=0x0200, CONNECT_STAY_AWAKE=0x0400, CONNECT_SAFE_MODE=0x0800, /* AP configuration flags */ AP_NO_DISASSOC_UPON_DEAUTH=0x10000, AP_HOSTPAL_SUPPORT=0x20000
)	
Description	Set the connect control flags.	
Return value	A_OK / A_ERROR	

7.4.44 qcom_op_set_mode

Definition	Set operation mode.	
Prototype	A_STATUS qcom_op_set_mode (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 mode</i>	0 = Station; 1 = AP
)	
Return value	A_OK / A_ERROR	

7.4.45 qcom_op_get_mode

Definition	Get operation mode.	
Prototype	A_STATUS qcom_op_get_mode (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 *pmode</i>	A pointer to operation mode
)	
Return value	A_OK / A_ERROR	

7.4.46 qcom_disconnect

Definition	Disconnect from the currently associated AP, or abort the current connection process.	
Prototype	A_STATUS qcom_disconnect (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
)	
Return value	A_OK / A_ERROR	

7.4.47 qcom_get_disconnect_reason

Definition	Get disconnect reason code.	
Prototype	A_STATUS qcom_get_disconnect_reason (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *preason	Disconnect reason code
)	
Return value	0x0 = Connected state 0x01 = NO_NETWORK_AVAIL 0x02 = LOST_LINK (bmiss) 0x03 = DISCONNECT_CMD 0x04 = BSS_DISCONNECTED 0x05 = AUTH_FAILED 0x06 = ASSOC_FAILED 0x07 = NO_RESOURCES_AVAIL 0x08 = CSERV_DISCONNECT 0x0A = INVALID_PROFILE 0x0B = DOT11H_CHANNEL_SWITCH 0x0C = PROFILE_MISMATCH 0x0D = CONNECTION_EVICTED 0x0E = IBSS_MERGE 0x0F = EXCESS_TX_RETRY (TX frames failed after maximum retries) 0x10 = SEC_HS_TO_RECV_M1 (Security 4-way handshake timed out waiting for M1) 0x11 = SEC_HS_TO_RECV_M3 (Security 4-way handshake timed out waiting for M3) 0x12 = TKIP_COUNTERMEASURES 0xFD = CCX_TARGET_ROAMING_INDICATION (Hack for CCX AP-Assisted roaming) 0xFE = CCKM_ROAMING_INDICATION (Hack for CCKM fast roaming)	

7.4.48 qcom_set_ap_country

Definition	Set beacon and probe response management frame countrycode. countrycode FF means disable country information element.	
Prototype	A_STATUS qcom_set_ap_country (
	A_UINT8 device_id,	Interface ID (default 0)
	A_CHAR *pcountry_code	A pointer to country code.
)	
Return value	A_OK / A_ERROR	

7.4.49 qcom_get_country_code

Definition	Get the country code in use.	
Prototype	A_STATUS qcom_get_country_code (
	A_UINT8 device_id,	Interface ID (default 0)
	A_CHAR *pcountry_code	A pointer to country code.
)	
Return value	A_OK / A_ERROR	

7.4.50 qcom_set_phy_mode

Definition	Set PHY operating mode.	
Prototype	A_STATUS qcom_set_phy_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT8 phymode	PHY modes: QCOM_11A_MODE = 0x1, QCOM_11B_MODE = 0x2, QCOM_11G_MODE = 0x3, QCOM_11N_MODE = 0x4, QCOM_HT40_MODE = 0x5,
)	
Return value	A_OK / A_ERROR	

7.4.51 qcom_get_phy_mode

Definition	Get PHY operating mode.	
Prototype	A_STATUS qcom_get_phy_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT8 *pphymode	A pointer to PHY mode
)	
Return value	A_OK / A_ERROR	

7.4.52 qcom_set_channel

Definition	Set channel ID.	
Prototype	A_STATUS qcom_set_channel (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT16 channel	Channel number
)	
Return value	A_OK / A_ERROR	

7.4.53 qcom_get_channel

Definition	Get the current channel.	
Prototype	A_STATUS qcom_get_channel (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT16 *pchannel	A pointer to channel
)	
Return value	A_OK / A_ERROR	

7.4.54 qcom_set_tx_power

Definition	Set Tx power level.	
Prototype	A_STATUS qcom_set_tx_power (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 dbm	Desired transmit power (1-17 dBm)
)	
Return value	A_OK / A_ERROR	

7.4.55 qcom_get_tx_power

Definition	Get Tx power level.	
Prototype	A_STATUS qcom_get_tx_power (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *pdbm	A pointer to Tx power
)	
Return value	A_OK / A_ERROR	

7.4.56 qcom_allow_aggr_set_tid

Definition	Set TID to enable/disable aggregation.	
Prototype	A_STATUS qcom_allow_aggr_set_tid(
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT16 tx_allow_aggr,	16-bit mask. Bit position indicates TID. 1 = Allow aggregation at uplink.
	A_UINT16 rx_allow_aggr	16-bit mask. Bit position indicates TID. 1 = Allow aggregation at downlink.
)	
Return value	A_OK / A_ERROR	

7.4.57 qcom_allow_aggr_get_tid

Definition	Get TID aggregation settings.	
Prototype	A_STATUS qcom_allow_aggr_get_tid(
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT16 *ptx_allow_aggr,	A pointer to Tx aggregation bit mask
	A_UINT16 *prx_allow_aggr	A pointer to Rx aggregation bit mask
)	
Return value	A_OK / A_ERROR	

7.4.58 qcom_scan_set_mode

Definition	Set firmware scan behavior by enabling foreground or background scanning.	
Prototype	A_STATUS qcom_scan_set_mode (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)

	<i>A_UINT32 mode</i>	1 = Foreground 0 = Background
)	
Return value	A_OK / A_ERROR	

7.4.59 qcom_scan_get_mode

Definition	Get the scan mode.	
Prototype	A_STATUS qcom_scan_get_mode (
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1 (default 0)
	<i>A_UINT32 *pmode</i>	A pointer to scan mode
)	
Return value	A_OK / A_ERROR	

7.4.60 qcom_get_state

Definition	Get chip Wi-Fi link state.	
Prototype	A_STATUS qcom_get_state (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT8 *pstate</i>	Chip states: K_WLAN_LINK_STATE_DISCONNECTED_STATE K_WLAN_LINK_STATE_STARTING_STATE K_WLAN_LINK_STATE_CONNECTING_STATE K_WLAN_LINK_STATE_AUTHENTICATING_STATE K_WLAN_LINK_STATE_CONNECTED_STATE K_WLAN_LINK_STATE_DISCONNECTING_STATE K_WLAN_LINK_STATE_WEP_KEY_NOT_MATCH
)	
Return value	A_OK / A_ERROR	

7.4.61 qcom_sec_set_wepkey

Definition	Set WEP encryption key by index.	
Prototype	A_STATUS qcom_sec_set_wepkey (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 keyindex,</i>	WEP key index (0-3)
	<i>A_CHAR *pkey,</i>	WEP key (format "ABCDEF1234")
)	
Return value	A_OK / A_ERROR	

7.4.62 qcom_sec_get_wepkey

Definition	Get WEP encryption key by index.	
Prototype	A_STATUS qcom_sec_get_wepkey (

	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 keyindex,</i>	WEP key index (0-3)
	<i>A_CHAR *pkey,</i>	A pointer to WEP key
)	
Return value	A_OK / A_ERROR	

7.4.63 qcom_sec_set_wepkey_index

Definition	Set WEP key index.	
Prototype	A_STATUS qcom_sec_set_wepkey_index (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 keyindex,</i>	WEP key index (0-3)
)	
Return value	A_OK / A_ERROR	

7.4.64 qcom_sec_get_wepkey_index

Definition	Get WEP key index.	
Prototype	A_STATUS qcom_sec_get_wepkey_index (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 keyindex,</i>	WEP key index (0-3)
)	
Return value	A_OK / A_ERROR	

7.4.65 qcom_sec_set_auth_mode

Definition	Set authentication mode.	
Prototype	A_STATUS qcom_sec_set_auth_mode (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT32 mode</i>	Authentication modes: WLAN_AUTH_NONE WLAN_AUTH_WPA WLAN_AUTH_WPA2 WLAN_AUTH_WPA_PSK WLAN_AUTH_WPA2_PSK WLAN_AUTH_WPA_CCKM WLAN_AUTH_WPA2_CCKM WLAN_AUTH_WPA2_PSK_SHA256 WLAN_AUTH_INVALID
)	
Return value	A_OK / A_ERROR	

7.4.66 qcom_sec_get_auth_mode

Definition	Get authentication mode.	
Prototype	A_STATUS qcom_sec_get_auth_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *pmode	A pointer to authentication mode
)	
Return value	A_OK / A_ERROR	

7.4.67 qcom_sec_set_encrypt_mode

Definition	Set encryption mode.	
Prototype	A_STATUS qcom_sec_set_encrypt_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 mode	Encryption modes: WLAN_CRYPT_NONE WLAN_CRYPT_WEP_CRYPT WLAN_CRYPT_TKIP_CRYPT WLAN_CRYPT_AES_CRYPT WLAN_CRYPT_BIP_CRYPT WLAN_CRYPT_KTK_CRYPT WLAN_CRYPT_INVALID
)	
Return value	A_OK / A_ERROR	

7.4.68 qcom_sec_get_encrypt_mode

Definition	Get encryption mode.	
Prototype	A_STATUS qcom_sec_get_encrypt_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *pmode	A pointer to encryption mode
)	
Return value	A_OK / A_ERROR	

7.4.69 qcom_sec_set_passphrase

Definition	Set passphrase.	
Prototype	A_STATUS qcom_sec_set_passphrase (
	A_UINT8 device_id,	Interface ID (default 0)
	A_CHAR *passphrase,	A pointer to passphrase
)	
Return value	A_OK / A_ERROR	

7.4.70 qcom_sec_get_passphrase

Definition	Get passphrase.	
Prototype	A_STATUS qcom_sec_get_passphrase (
	A_UINT8 device_id,	Interface ID (default 0)
	A_CHAR *passphrase,	A pointer to passphrase
)	
Return value	A_OK / A_ERROR	

7.4.71 qcom_power_set_mode

Definition	Set power mode.	
Prototype	A_STATUS qcom_power_set_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 powerMode	1 = REC_POWER. Conserve power. 2 = MAX_PERF_POWER. Maximum performance. Other values = Reserved
)	
Description	<p>In the conserve power mode (recommended), the device enters sleep state during idle period to conserve power. Performance is impacted by this mechanism because it takes up to 2 ms to resume operation after exiting sleep state.</p> <p>In the maximum performance mode, the device never enters sleep state (no need to wakeup) to achieve better performance at the cost of higher power consumption.</p>	
Return value	A_OK / A_ERROR	

7.4.72 qcom_power_get_mode

Definition	Get power mode.	
Prototype	A_STATUS qcom_power_get_mode (
	A_UINT8 device_id,	Interface ID (default 0)
	A_UINT32 *powerMode	A pointer to power mode
)	
Description	<p>In the conserve power mode (recommended), the device enters sleep state during idle period to conserve power. Performance is impacted by this mechanism because it takes up to 2 ms to resume operation after exiting sleep state.</p> <p>In the maximum performance mode, the device never enters sleep state (no need to wakeup) to achieve better performance at the cost of higher power consumption.</p>	
Return value	A_OK / A_ERROR	

7.4.73 qcom_suspend_enable

Definition	Enable suspend function.	
Prototype	A_STATUS qcom_suspend_enable (
	A_BOOL enable	1 = Enable; 0 = Disable.

)	
Return value	A_OK / A_ERROR	

7.4.74 qcom_suspend_start

Definition	Start the suspend function and restore after the specified time.	
Prototype	A_STATUS qcom_suspend_start (
	<i>A_UINT32 time</i>	Suspend period in seconds
)	
Return value	A_OK / A_ERROR	

7.4.75 qcom_power_set_parameters

Definition	Set power parameters.	
Prototype	A_STATUS qcom_power_set_parameters (
	<i>A_UINT8 device_id,</i>	Interface ID (default 0)
	<i>A_UINT16 idlePeriod,</i>	Period in ms for the device to remain awake after frame transmit/receive is complete and before entering SLEEP state
	<i>A_UINT16 psPollNum,</i>	The number of PS-poll messages sent by the device before notifying the AP it is awake
	<i>A_UINT16 dtimPolicy,</i>	WMI DTIM policy 1 = Ignore DTIM. The device does not listen to any contents after Beacon (CAB). 2 = Normal DTIM. DTIM period follows the listen interval. For example, if the listen interval is 4 and the DTIM period is 2, the device wakes up every fourth beacon. 3 = Stick DTIM. The device attempts to receive all CAB traffic. For example, if the DTIM period is 2 and the listen interval is 4, the device wakes up every second beacon. 4 = Auto DTIM (decided by device)
	<i>A_UINT16 tx_wakeup_policy,</i>	WMI Tx wakeup policy 1 = Wakeup upon uplink traffic. The number of uplink frames in a Beacon interval to trigger the transition to awake is determined by NUM_TX_TO_WAKEUP. 2 = Not wakeup upon uplink traffic
	<i>A_UINT16 num_tx_to_wakeup,</i>	
	<i>A_UINT16 ps_fail_event_policy</i>	Power save fail event policy 1 = Any null frame with PM = 1. Tx failure is notified by the WMI_TARGET_PM_ERR_FAIL event. 2 = Ignore null frame with PM = 1 (Tx failures during scan).
)	
Return value	A_OK / A_ERROR	

7.4.76 qcom_promiscuous_enable

Definition	Enable promiscuous mode.	
Prototype	A_STATUS qcom_promiscuous_enable (
	A_BOOL promiscuousEn,	TRUE = Enable promiscuous mode. FALSE = Disable promiscuous mode.
)	
Return value	A_OK / A_ERROR	

7.4.77 qcom_set_promiscuous_rx_cb

Definition	Set the call back function for promiscuous mode.	
Prototype	qcom_set_promiscuous_rx_cb (
	WLAN_PROMISCUOUS_CB cb,	typedef void (*WLAN_PROMISCUOUS_CB)(unsigned char *buf, int length); cb is a hook function to process received packets of type WLAN_PROMISCUOUS_CB. Since the hook function runs in the target domain, users must not call ThreadX API or Driver APIs in hook function.
)	
Return value	A_OK	

7.4.78 qcom_get_bssid

Definition	Get the BSSID of the device context for the given device ID.	
Prototype	A_STATUS qcom_get_bssid (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	A_UINT8 pmode[ATH_MAC_LEN]	Pointer to the memory where the BSSID should be copied to.
)	
Return value	A_OK / A_ERROR	

7.4.79 qcom_scan_params_set

Definition	Set scan parameters.	
Prototype	A_STATUS qcom_scan_param_set (
	A_UINT8 device_id,	Interface ID

	<pre> qcom_scan_params_t *pParam </pre>	<pre> typedef struct _qcom_scan_params { A_UINT16 fgStartPeriod; //First interval in seconds when //the device disconnects from an AP or receives a //connect command. 0 = Use the reset value, 0xFFFF = //Disable foreground scan. A_UINT16 fgEndPeriod; //Maximum interval in seconds //the device waits between foreground scans. Range: //ForegroundScanStartPeriod to 65535. 0 = Use reset //value (default). A_UINT16 bgPeriod; //Period in seconds of background //scan. 0xFFFF = Disable background scan. Default = 60. A_UINT16 maxActChDwellTimeInMs; //Period in msec the //device stays on a particular channel during active //scanning. 0 = Use the reset value. Default = 20 A_UINT16 pasChDwellTimeInMs; // Period in msec the //device stays on a particular channel during passive //scanning. 0 = Use the reset value. Default = 50 A_UINT8 shortScanRatio; //Number of short scans to //perform for each long scan. Default = 3 A_UINT8 scanCtrlFlags; //0x1 = Set whether the //QCA4010/QCA4012 can scan using the connect //command, else no probe request is sent during connect; //0x2 = Set whether the QCA4010/QCA4012 can scan for //the SSID it will connect to or is already connected to; //0x4 = Set whether active scan is enabled for the SSID; //0x08 = Set whether roam scan is enabled when BMISS //and a low RSSI event happens; 0x10 = Set whether //follows customer BSSINFO reporting rule; 0x20 = If not //set, the target does not scan automatically to find an AP //after a disconnect event. 0x40 = A scan complete event //with canceled status is generated when a scan is //preempted before completion. 0x0 = Disable all scan //control flags; 0xFF = Leave the current settings. Default // = 0x2f. A_UINT16 minActChDwellTimeInMs; //Default = 20 A_UINT16 maxActScanPerSsid; //Default = 1 A_UINT32 maxDfsChActTimeInMs; //Default = 2000 } qcom_scan_params_t; </pre>
)	
Return value	A_OK on success, A_ERROR on failure	

7.4.80 qcom_param_set

Definition	Set the PARAM_GROUP_SYSTEM.	
Prototype	A_STATUS qcom_param_set (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	A_UINT16 grp_id,	Param group ID: 1: system 2: wlan 3: network

	<i>A_UINT16 param_id,</i>	Param group timeout, relative with grp_id. When grp_id=1: 1: system sleep maintenance timeout 2: system sleep watchdog timeout When grp_id=3: 1: network dns timeout
	<i>Void * data,</i>	Timeout value
	<i>A_UINT8 data_length</i>	The length of timeout value
	<i>A_BOOL wait_for_status</i>	Should be always true currently
)	
Return value	A_OK / A_ERROR	

7.4.81 qcom_mcast_filter_enable

Definition		
Prototype	A_STATUS qcom_mcast_filter_enable (
	<i>A_UINT8 enable</i>	Enable/disable multicast, 0 or 1 (default 1)
)	
Return value	A_OK / A_ERROR	

7.4.82 qcom_set_bmiss_time

Definition	Configure bmiss threshold for STA mode. Of parameters bmiss_num and bmiss_time, either one shall be provided (that is, a nonzero value). If both are provided, bmiss_num will take effect and bmiss_time won't.	
Prototype	A_STATUS qcom_set_bmiss_time(
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1
	<i>A_UINT16 bmiss_num,</i>	Threshold in number of beacons
	<i>A_UINT16 bmiss_time</i>	Threshold in unit of TU (1 TU equals 1024 microsecond)
)	
Return value	A_OK / A_EINVAL	

7.4.83 qcom_set_vendor_specific_ie_cmd

Definition	Set vendor specific information elements at the end of management frame (beacon, probe req, probe resp, assoc req).	
Prototype	A_STATUS qcom_set_vendor_specific_ie_cmd (
	<i>A_UINT8 device_id,</i>	Interface ID, 0 or 1

	<i>A_UINT8 mgmt_frm_type</i>	Management frame type: WMI_FRAME_BEACON = 0, WMI_FRAME_PROBE_REQ = 1, WMI_FRAME_PROBE_RESP = 2, WMI_FRAME_ASSOC_REQ = 3
	<i>A_UINT8 *ie_info</i>	Pointer to Information elements memory buffer
	<i>A_UINT16 len</i>	The length of information elements. Maximum length is 255
)	
Description	Set vendor specific information elements in the end of management frame. If ie_info = NULL and len = 0, delete the old vendor specific information elements. This API supports multiple IEs, and the IEs need to be filled in one memory buffer. The len is the total length of IEs.	
Return value	A_OK on success, A_NO_MEMORY on failure when there is no memory.	

7.4.84 qcom_ani_enable

Definition	Enable or disable ANI.	
Prototype	void qcom_ani_enable(
	<i>A_BOOL enable</i>	TRUE = enable; FALSE = disable
)	
Return value	N/A	

7.5 WPS APIs

This section provides brief description of the APIs exported by the Qualcomm Technologies library that can be used by the applications to perform WPS operations.

7.5.1 WPS data structure

This section describes various data structures exported by the Qualcomm Technologies library to perform WPS operations by applications.

qcom_wps_event_t

This is a generic structure which contains union of multiple structures. Based on the event received, the payload is interpreted using appropriate structure in the union. The generic structure definition of WPS event payload is as follows:

```
typedef struct {
    union {
        qcom_wps_profile_info_t profile_info;
        /* Add more events here if needed */
    } event;
} qcom_wps_event_t;
```

Details about the event-specific structures are provided in subsequent sections.

qcom_wps_profile_info_t

This structure contains the information about the WPS credentials that are derived after an 8-way WPS handshake. This structure should be used by the application to parse the payload on reception of the QCOM_WPS_PROFILE_EVENT event from the lower layers. The structure information is as follows:

```
typedef struct qcom_wps_profile_info_s
{
    QCOM_WPS_ERROR_CODE    error_code; /* WPS_ERROR_CODE */
    qcom_wps_credentials_t credentials ;
} qcom_wps_profile_info_t;
```

QCOM_WPS_ERROR_CODE

This is an enumeration of all the possible error codes that can be obtained at the end of WPS handshake. Possible values for this field are listed below.

```
/* WPS Error codes */
typedef enum {
    QCOM_WPS_STATUS_SUCCESS = 0,
    QCOM_WPS_ERROR_INVALID_START_INFO = 0x1,
    QCOM_WPS_ERROR_MULTIPLE_PBC_SESSIONS,
    QCOM_WPS_ERROR_WALKTIMER_TIMEOUT,
    QCOM_WPS_ERROR_M2D_RCVD,
    QCOM_WPS_ERROR_PWD_AUTH_FAIL,
    QCOM_WPS_ERROR_CANCELLED,
    QCOM_WPS_ERROR_INVALID_PIN
} QCOM_WPS_ERROR_CODE;
```

qcom_wps_credentials_t

This parameter contains the actual credential information. The below table describes various parameters that are received as part of this event.

Data type	Member name	Description
A_UINT16	ap_channel	Channel of the peer in MHz
A_UINT8	ssid	SSID string of the network. Maximum length is 32.
A_UINT8	ssid_len	Length of the SSID string
WLAN_AUTH_MODE	auth_type	Type of authentication used <ul style="list-style-type: none"> ▪ WLAN_AUTH_NONE ▪ WLAN_AUTH_WPA ▪ WLAN_AUTH_WPA2 ▪ WLAN_AUTH_WPA_PSK ▪ WLAN_AUTH_WPA2_PSK ▪ WLAN_AUTH_WPA_CCKM ▪ WLAN_AUTH_WPA2_CCKM ▪ WLAN_AUTH_WPA2_PSK_SHA256 ▪ WLAN_AUTH_INVALID

Data type	Member name	Description
WLAN_CRYPT_MODE	encr_type	Type of encryption used <ul style="list-style-type: none"> ▪ WLAN_CRYPT_NONE ▪ WLAN_CRYPT_WEP_CRYPT ▪ WLAN_CRYPT_TKIP_CRYPT ▪ WLAN_CRYPT_AES_CRYPT ▪ WLAN_CRYPT_BIP_CRYPT ▪ WLAN_CRYPT_KTK_CRYPT ▪ WLAN_CRYPT_INVALID
A_UINT8	key_idx	Key index used
A_UINT8	key	Passphrase of the network. Maximum length is 64.
A_UINT8	key_len	Length of the passphrase
A_UINT8	mac_addr	MAC address of the Peer

7.5.2 qcom_wps_enable

Definition	Enable/ disable WPS function. This function must be invoked along with qcom_wps_resister_event_handler().	
Prototype	void qcom_wps_enable (
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	int enable	1 = Enable; 0 = Disable
)	
Return value	N/A	

7.5.3 qcom_wps_start

Definition	Start WPS connection.	
Prototype	void qcom_wps_start(
	A_UINT8 device_id,	Interface ID, 0 or 1 (default 0)
	int connect,	1 = After WPS procedure is finished, if the device is in enrollee mode, it connects registrar immediately. 0 = No action after WPS procedure is finished.
	int mode,	0 = PIN mode; 1 = PBC mode
	char *pin	The address of PIN string when mode is PIN mode.
)	
Return value	N/A	

7.5.4 qcom_wps_stop

Definition	Stop WPS connection.	
Prototype	void qcom_wps_stop (
	A_UINT8 device_id	Interface ID, 0 or 1
)	
Return value	N/A	

7.5.5 qcom_wps_connect

Definition	Initiate WPS connection on the specific interface ID.	
Prototype	void qcom_wps_connect (
	<i>A_UINT8 device_id</i>	Interface ID, 0 or 1
)	
Return value	A_OK on success, A_ERROR on error	

7.5.6 qcom_wps_register_event_handler

Definition	Allow application to register a callback with WPS to indicate WPS error status and take necessary action for the event received. This function must be invoked whenever qcom_wps_enable() is called.	
Prototype	void qcom_wps_register_event_handler (
	<i>qcom_wps_event_handler_t handler</i>	Event handler function pointer. The handler prototype information is provided in the WPS data structure section.
	void *pArg	Application context pointer to be supplied in the callback
)	
Return value	N/A	

7.5.7 qcom_wps_set_credentials

Definition	Set the WPS credentials received over the WPS profile event.	
Prototype	A_STATUS qcom_wps_set_credentials (
	<i>A_UINT8 device_id</i>	Interface ID, 0 or 1
	<i>qcom_wps_credentials_t *pwps_prof</i>	Credentials structure pointer. Details of this data structure are provided in WPS data structure section
)	
Return value	A_OK on Success, A_ERROR on error	

7.5.8 qcom_wps_event_handler_t

Definition	The prototype of the callback function that applications should register with the WPS library. This function is invoked by WPS library on receiving WPS events from the target.	
Prototype	void qcom_wps_event_handler_t (
	<i>A_UINT8 device_ID</i>	Device ID for which the event received is applicable

	<i>QCOM_WPS_EVENT_TYPE</i> <i>uEventID</i>	Identifier which identifies the event type from enumerations of data type <i>QCOM_WPS_EVENT_TYPE</i> . The Event ID information is as follows. <i>QCOM_WPS_PROFILE_EVENT</i> : indicated by the target on successful completion of WPS handshake.
	<i>qcom_wps_event_t</i> * <i>pEvtBuffer</i>	Payload the event received. More details about this structure are provided under WPS Data structures section
	<i>void</i> * <i>pArg</i>	User-specified context provided during the registration of callback
)	
Return value	N/A	

7.6 P2P APIs

7.6.1 qcom_p2p_func_init

Definition	Enable/disable P2P function.	
Prototype	<i>void</i> <i>qcom_p2p_func_init</i> (
	<i>A_UINT8</i> <i>device_id</i> ,	Interface ID (default 0)
	<i>A_INT32</i> <i>enable</i>	1 = Enable; 0 = Disable
)	
Return value	N/A	

7.6.2 qcom_p2p_func_cancel

Definition	Cancel the P2P function.	
Prototype	<i>A_STATUS</i> <i>qcom_p2p_func_cancel</i> (
	<i>A_UINT8</i> <i>device_id</i>	Interface ID (default 0)
)	
Return value	<i>A_OK</i> on success, error code on error	

7.6.3 qcom_p2p_func_set_pass_ssid

Definition	Set the passphrase and ssid for P2P.	
Prototype	<i>A_STATUS</i> <i>qcom_p2p_func_set_pass_ssid</i>	
	<i>A_UINT8</i> <i>device_id</i>	Interface ID (default 0)
	<i>A_CHAR</i> * <i>ppass</i>	The point of passphrase
	<i>A_CHAR</i> * <i>pssid</i>	The point of SSID
)	
Return value	<i>A_OK</i> on success, error code on error	

7.6.4 qcom_p2p_func_get_pass_ssid

Definition	Get the passphrase and ssid for P2P.	
Prototype	A_STATUS qcom_p2p_func_get_pass_ssid	
	A_UINT8 device_id	Interface ID (default 0)
	A_CHAR *ppass	The point of passphrase
	A_CHAR *pssid	The point of SSID
)	
Return value	A_OK on success, error code on error	

7.6.5 qcom_p2p_func_auth

Definition	Authorize the peer P2P device to connect itself.	
Prototype	A_STATUS qcom_p2p_func_auth (
	A_UINT8 device_id	Interface ID (default 0)
	A_INT32 dev_auth	0 = Authorize; 1 = De-authorize
	enum p2p_wps_method wps_method	WPS method: P2P_WPS_NOT_READY P2P_WPS_PIN_LABEL P2P_WPS_PIN_DISPLAY P2P_WPS_PIN_KEYPAD P2P_WPS_PBC
	A_UINT8 *peer_mac	The point of peer P2P device MAC address
	A_BOOL persistent	persistent group TRUE: persistent; FALSE: non-persistent
)	
Return value	A_OK on success, error code on error	

7.6.6 qcom_p2p_func_connect

Definition	Connect to the peer P2P device.	
Prototype	A_STATUS qcom_p2p_func_connect (
	A_UINT8 device_id	Interface ID (default 0)
	enum p2p_wps_method wps_method	WPS method: P2P_WPS_NOT_READY P2P_WPS_PIN_LABEL P2P_WPS_PIN_DISPLAY P2P_WPS_PIN_KEYPAD P2P_WPS_PBC
	A_UINT8 *peer_mac	The point of peer P2P device MAC address
	A_BOOL persistent	persistent group TRUE: persistent; FALSE: non-persistent
)	
Return value	A_OK on success, error code on error	

7.6.7 qcom_p2p_func_start_go

Definition	Start P2P GO (autonomous GO and after GO negotiation).	
Prototype	A_STATUS qcom_p2p_func_start_go (
	A_UINT8 device_id	Interface ID (default 0)
	A_UINT8 *pssid	Pointer to SSID
	A_UINT8 *ppass	Pointer to passphrase
	A_INT32 channel	GO operating channel
	A_BOOL persistent	persistent group TRUE: persistent; FALSE: non-persistent
)	
Return value	A_OK on success, error code on error	

7.6.8 qcom_p2p_func_set_config

Definition	Set the P2P configuration.	
Prototype	A_STATUS qcom_p2p_func_set_config (
	A_UINT8 device_id	Interface ID (default 0)
	A_UINT32 uigo_intend	Go intent value
	A_UINT32 uiclisten_ch	Listen channel
	A_UINT32 uiop_ch	Operation channel
	A_UINT32 uiage	Aging time
	A_UINT32 reg_class	Register domain
	A_UINT32 op_reg_class	Operating register domain
	A_UINT32 max_node_count	Maximum node number (up to 5)
)	
Return value	A_OK on success, error code on error	

7.6.9 qcom_p2p_func_invite

Definition	Invite the other P2P device to join the P2P group.	
Prototype	A_STATUS qcom_p2p_func_invite (
	A_UINT8 device_id	Interface ID (default 0)
	A_CHAR *pssid	The point of SSID
	enum p2p_wps_method wps_method	WPS method: P2P_WPS_NOT_READY P2P_WPS_PIN_LABEL P2P_WPS_PIN_DISPLAY P2P_WPS_PIN_KEYPAD P2P_WPS_PBC
	A_UINT8 *pmac	The point of P2P device invited MAC address
	A_BOOL persistent	persistent group TRUE: persistent; FALSE: non-persistent

	<i>P2P_INV_ROLE</i> role	P2P_INV_ROLE_GO, P2P_INV_ROLE_ACTIVE_GO, P2P_INV_ROLE_CLIENT,
)	
Return value	A_OK on success, error code on error	

7.6.10 qcom_p2p_func_join

Definition	Join the P2P group.	
Prototype	A_STATUS qcom_p2p_func_join (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>enum p2p_wps_method wps_method</i>	WPS method: P2P_WPS_NOT_READY P2P_WPS_PIN_LABEL P2P_WPS_PIN_DISPLAY P2P_WPS_PIN_KEYPAD P2P_WPS_PBC
	<i>A_UINT8 *pmac</i>	The point of P2P device invited MAC address
	<i>char *ppin</i>	The point of pin string
)	
Return value	A_OK on success, error code on error	

7.6.11 qcom_p2p_func_prov

Definition	Start P2P provision negotiation to the peer P2P device.	
Prototype	A_STATUS qcom_p2p_func_prov (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>enum p2p_wps_method wps_method</i>	WPS method: WPS_NOT_READY WPS_PIN_LABEL WPS_PIN_DISPLAY WPS_PIN_KEYPAD WPS_PBC
	<i>A_UINT8 *pmac</i>	The point of P2P device invited MAC address
)	
Return value	A_OK on success, error code on error	

7.6.12 qcom_p2p_func_set_opps

Definition	Set the OppPS parameters for power save mode.	
Prototype	A_STATUS qcom_p2p_func_set_opps (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>A_UINT8 enable</i>	1 = Enable; 0 = Disable

	<i>A_UINT8 ctwin</i>	Client traffic windows
)	
Return value	A_OK on success, error code on error	

7.6.13 qcom_p2p_func_set_noa

Definition	Set NOA parameters for power save mode.	
Prototype	A_STATUS qcom_p2p_func_set_noa (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>A_UINT8 uccount</i>	The number of absence intervals
	<i>A_UINT32 uistart</i>	The start time for the schedule
	<i>A_UINT32 uiduration</i>	The maximum duration
	<i>A_UINT32 uiinterval</i>	The length of the NOA interval
)	
Return value	A_OK on success, error code on error	

7.6.14 qcom_p2p_func_find

Definition	Start to find other P2P devices.	
Prototype	A_STATUS qcom_p2p_func_find (
	<i>A_UINT8 device_id</i> ,	Interface ID (default 0)
	<i>P2P_DEV_CTXT *dev</i> ,	NULL, not useful
	<i>P2P_DISC_TYPE type</i> ,	Find type: P2P_DISC_START_WITH_FULL P2P_DISC_ONLY_SOCIAL P2P_DISC_PROGRESSIVE
	<i>A_UINT32 timeout</i>	Timeout time, unit is second
)	
Return value	A_OK on success, error code on error	

7.6.15 qcom_p2p_func_stop_find

Definition	Stop P2P find action.	
Prototype	A_STATUS qcom_p2p_func_stop_find (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
)	
Return value	A_OK on success, error code on error	

7.6.16 qcom_p2p_func_get_node_list

Definition	Get the P2P node list. This API is blocking and should be called in the context of an application thread.	
Prototype	A_STATUS qcom_p2p_func_get_node_list (

	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>P2P_NODE_LIST_EVENT *app_buf</i>	Application buffer to hold node list
	<i>A_UINT32 buf_size</i>	Application buffer size
)	
Return value	A_OK on success, error code on error	

7.6.17 qcom_p2p_func_get_network_list

Definition	Get the P2P persistent network list. This API is blocking and should be called in the context of an application thread.	
Prototype	A_STATUS qcom_p2p_func_get_network_list (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>P2P_PERSISTENT_MAC_LIST*app_buf</i>	Application buffer to hold node list
	<i>A_UINT32 buf_size</i>	Application buffer size
)	
Return value	A_OK on success, error code on error	

7.6.18 qcom_p2p_func_set

Definition	Set the P2P configuration	
Prototype	A_STATUS qcom_p2p_func_set (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>P2P_DEV_CTXT *dev</i>	NULL, not useful
	<i>P2P_CONF_ID config_id</i>	Config ID: P2P_CONFIG_LISTEN_CHANNEL P2P_CONFIG_CROSS_CONNECT P2P_CONFIG_SSID_POSTFIX P2P_CONFIG_INTRA_BSS P2P_CONFIG_CONCURRENT_MODE P2P_CONFIG_GO_INTENT P2P_CONFIG_DEV_NAME P2P_CONFIG_P2P_OPMODE
	<i>void *data</i>	The point of specific data
	<i>A_UINT32 data_length</i>	The length of data
)	
Return value	A_OK on success, error code on error	

7.6.19 qcom_p2p_func_invite_auth

Definition	Determine join or reject to join the P2P group.	
Prototype	A_STATUS qcom_p2p_func_invite_auth (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)

	<i>P2P_FW_INVITE_REQ_RSP_CMD *inv</i>	<i>typedef PREPACK struct { A_UINT16 force_freq; A_UINT8 status; A_UINT8 dialog_token; A_UINT8 is_go; A_UINT8 group_bssid[MAC_LEN]; } P2P_FW_INVITE_REQ_RSP_CMD;</i>
)	
Return value	A_OK on success, error code on error	

7.6.20 qcom_p2p_func_wps_start_no_scan

Definition	Start WPS for client after GO negotiation is complete.	
Prototype	A_STATUS _qcom_p2p_func_wps_start_no_scan (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>A_BOOL is_go</i>	TRUE: GO; FALSE: Client
	<i>A_BOOL is_push_method</i>	TRUE: WPS PBC; FALSE: Other
	<i>A_UINT32 channel</i>	Channel on which to start WPS
	<i>A_UINT8 *pssid</i>	Pointer to SSID
	<i>A_UINT8 *pin</i>	Pointer to PIN
	<i>A_UINT8 *pmac</i>	Pointer to MAC address
)	
Return value	A_OK on success, error code on error	

7.6.21 qcom_p2p_func_listen

Definition	Have the P2P device listen to the listen channel.	
Prototype	A_STATUS _qcom_p2p_func_listen (
	<i>A_UINT8 device_id</i>	Interface ID (default 0)
	<i>A_UINT32 timeout</i>	Timeout in seconds
)	
Return value	A_OK on success, error code on error	

7.6.22 qcom_p2p_get_role

Definition	Get the device role from WMI event buffer.	
Prototype	P2P_INV_ROLE qcom_p2p_func_get_role (
	<i>A_UINT32 role</i>	Role in WMI event buffer
)	
Return value	P2P_INV_ROLE	

7.6.23 qcom_p2p_event_handler

Definition	The prototype of the callback function that applications should register with the P2P library. This function is invoked by P2P library on receiving P2P events from the target.	
Prototype	void qcom_p2p_event_handler (
	A_UINT8 device_ID	Device ID for which the event received is applicable
	P2P_EVENT_ID event_id	Identifier which identifies the event type from enumerations of data type P2P_EVENT_ID.
	A_UINT8 *pEvtBuffer	Payload the event received
	void *pArg	User-specified context provided during the registration of callback
)	
Return value	N/A	

7.6.24 qcom_p2p_func_register_event_handler

Definition	Register application handler for P2P WMI events.	
Prototype	A_STATUS qcom_p2p_func_register_event_handler (
	qcom_p2p_event_handler cb	Pointer to event handler in application
	void *arg	Argument to be passed when event handler is called
)	
Return value	A_OK on success, error code on error	

7.7 SSL APIs

7.7.1 qcom_SSL_CTX_new

Definition	Create an SSL context for use by one SSL session.	
Prototype	SSL_CTX* qcom_SSL_CTX_new (
	SSL_ROLE_T role,	Role = 1 (Server); 2 (Client)
	A_INT32 inbufSize,	Initial input buffer size
	A_INT32 outbufSize,	Initial output buffer size
	A_INT32 reserved	
)	
Description	This function must be called before using any other SSL function.	
Return value	Pointer to the new context on success; null on error	

7.7.2 qcom_SSL_CTX_free

Definition	Release SSL context.	
Prototype	void qcom_SSL_CTX_free (
	SSL_CTX * ctx	SSL context to release
)	
Return value	SSL_CLNT_OK on success; error code on error	

7.7.3 qcom_SSL_new

Definition	Create an SSL connection object for use with an SSL session.	
Prototype	SSL* qcom_SSL_new (
	SSL_CTX * ctx	Pointer to SSL context
)	
Description	Close the connection object with qcom_SSL_shutdown after transaction is completed.	
Return value	Pointer to SSL structure on success; null on error	

7.7.4 qcom_SSL_accept

Definition	Accept an SSL session connection request from a remote client application.	
Prototype	int qcom_SSL_accept (
	SSL * ssl	Pointer to SSL structure
)	
Return value	1 on success; error code on error	

7.7.5 qcom_SSL_connect

Definition	Start an SSL session with a remote SSL server.	
Prototype	int qcom_SSL_connect (
	SSL * ssl	Pointer to SSL structure
)	
Return value	1 on success; error code on error	

7.7.6 qcom_SSL_configure

Description	Configure the SSL connection.	
Prototype	A_INT32 qcom_SSL_configure(
	SSL *ssl,	SSL connection
	SSL_CONFIG *cfg	Pointer to the structure that contains the SSL configuration
)	
Description	Provide information on Cipher, verification policy, and so forth.	
Return value	1 on success, negative error code on error	

7.7.7 qcom_SSL_context_configure

Definition	Configure the SSL context. All SSL connections using this SSL context can use the provided configuration.	
Prototype	A_INT32 qcom_SSL_context_configure (
	SSL_CTX* <i>ss/Ctx</i> ,	SSL context
	SSL_CONFIG * <i>cfg</i>	Pointer to the structure that contains the SSL configuration
)	
Description	Provide information on cipher, verification policy, and so forth.	
Return value	1 on success, negative error code on error	

7.7.8 qcom_SSL_setCaList

Description	Set CA list for the SSL object	
Prototype	A_INT32 qcom_SSL_setCaList(
	SSL_CTX * <i>ctx</i> ,	SSL context
	A_UINT8* <i>cert</i> ,	Reference to SSL CA list
	A_UINT32 <i>size</i>	Size of the array
)	
Description	SSL performs certificate validation based on the peer certificate. Only one CA list is allowed, thus the CA list must include all root certificates required for the session.	
Return value	1 on success, negative error code on error	

7.7.9 qcom_SSL_addCert

Description	Add a certificate to the SSL object.	
Prototype	A_INT32 qcom_SSL_addCert(
	SSL_CTX * <i>ctx</i> ,	Pointer to SSL context
	A_UINT8* <i>cert</i> ,	Address of array of binary data
	A_UINT32 <i>size</i>	Size of array
)	
Description	SSL object in server mode requires at least one certificate.	
Return value	1 on success, negative error code on error	

7.7.10 qcom_SSL_storeCert

Description	Store a certificate or CA list to flash.	
Prototype	A_INT32 qcom_SSL_storeCert(
	A_CHAR * <i>name</i> ,	Name of the certificate
	A_UINT8* <i>cert</i> ,	Address of array of binary data
	A_UINT32 <i>size</i>)	Size of array

)	
Return value	1 on success, negative error code on error	

7.7.11 qcom_SSL_loadCert

Definition	Load certificate from flash	
Prototype	A_INT32 qcom_SSL_loadCert(
	SSL_CTX *ctx	Pointer to SSL Context
	SSL_CERT_TYPE_T type	type - Certificate or CA List
	Char *name)	Name of the certificate
)	
Description	Load a certificate or CA list from flash and store it in the SSL object.	
Return value	1 on success, negative error code on error	

7.7.12 qcom_SSL_listCert

Definition	List the certificates stored in the flash	
Prototype	A_INT32 qcom_SSL_listCert(
	SSL_FILE_NAME_LIST *fileNames)	Pointer to array containing names of certificate files stored in flash.
)	
Description	Return the list of certificates stored in the flash	
Return value	1 on success, negative error code on error	

7.7.13 qcom_SSL_read

Definition	Read application data from an SSL session.	
Prototype	int qcom_SSL_read (
	SSL *ssl,	Pointer to SSL structure
	char *buf,	Pointer to the buffer to read
	int num	Maximum bytes of data to read
)	
Return value	Byte count of the read data on success; error code on error	

7.7.14 qcom_SSL_set_fd

Definition	Assign a socket to an SSL session.	
Prototype	int qcom_SSL_set_fd (
	SSL *ssl,	Pointer to SSL session
	int fd	File descriptor of the socket
)	
Return value	1 on success; negative error code on error	

7.7.15 qcom_SSL_shutdown

Definition	Shut down data flow for an SSL session.	
Prototype	int qcom_SSL_shutdown (
	SSL * ssl,	Pointer to SSL session
)	
Return value	0 = The SSL_shutdown function is first issued from the application. Continue issuing the SSL_shutdown function until the code 1 is received, which means the remote application is shut down. 1 = Both client and server applications have issued the SSL_shutdown function (in SSL version 3 and TLS version 1). -1 = error	

7.7.16 qcom_SSL_write

Definition	Write application data over an SSL session.	
Prototype	int qcom_SSL_write (
	SSL * ssl,	Pointer to SSL session
	char * buf,	Pointer to the data to send
	int num	Number of bytes of data to send
)	
Return value	Byte count of the sent data on success; error code on error	

7.7.17 qcom_SSL_set_tm

Definition	Set time used in TLS certificate checking.	
Prototype	A_INT32 qcom_SSL_set_tm (
	tmx_t *ptm	Time to set
)	
Description	Set time used in TLS certificate checking to get out of the limitation of SNTP service.	
Return value	A_OK on success, A_ERROR on error	

7.8 Hardware APIs

7.8.1 qcom_gpio_apply_peripheral_configuration

Definition	This function obtains the GPIO pin map for the given peripheral ID using the GPIO active configuration set (for enabling) and GPIO inactive configuration set (for disabling) and configures the pins to the requested power state. If active configuration is not found for the requested peripheral ID, this API returns an error.	
Prototype	A_STATUS qcom_gpio_apply_peripheral_configuration(
	A_UINT32 peripheral_id,	Peripheral ID to be configured to a specific mode. Refer to qcom_gpio.h for the list of peripheral IDs.

	<i>A_BOOL low_power_mode</i>	Specifies the mode for the given peripheral ID. A_TRUE = Set the pins in inactive configuration. A_FALSE = Set the pins in active configuration.
)	
Return value	A_OK on success, A_ERROR or A_HW_CONFIG_ERROR on failure. If A_HW_CONFIG_ERROR is returned, the application or caller can invoke <code>qcom_gpio_peripheral_pin_conflict_check()</code> to get the conflicting peripherals that caused the error.	

7.8.2 qcom_gpio_add_alternate_configurations

Definition	This API allows applications to dynamically add multiple active configurations to the given GPIO pin. For a given pin, this API should be called only once with all possible active configuration at the time of application initialization. If this API is called more than once with same pin number, this API returns failure. During store-recall, the dynamic active configuration is saved and restored as part of the GPIO store-recall framework. If application attempts to add dynamic configuration during recall, this API returns failure	
Prototype	A_STATUS qcom_gpio_add_alternate_configurations(
	<i>A_UINT32 pin_num,</i>	Physical GPIO pin number to which the given dynamic configuration is to add.
	<i>A_UINT32 num_configs,</i>	Number of dynamic configurations to be added as part of pPinConfigs.
	<i>const gpio_pin_peripheral_config_t *pPinConfigs</i>	The array of structure of type <code>gpio_pin_peripheral_config_t</code> . Every index of the array represents various active configurations the application wants to support on the pin provided in the first parameter. Structure definition of <code>gpio_pin_peripheral_config_t</code> : typedef struct gpio_pin_peripheral_config_s { A_UINT32 peripheral_id; A_UINT32 gpio_active_config; } gpio_pin_peripheral_config_t;
)	
Return value	A_OK on success, A_ERROR on failure (for example, redundant peripheral IDs in the dynamic configuration for the given pin, multiple dynamic configurations added for the same pin)	

7.8.3 qcom_gpio_peripheral_pin_conflict_check

Definition	This API allows applications to obtain the peripheral map which are conflicting with the peripheral ID requested by the application. This API internally obtains GPIO map for requested peripheral ID and then does the bitwise OR of peripheral IDs that are active in pins obtained in GPIO map. If any failure occurs during peripheral enable/disable, initialization/de-initialization, or qcom_gpio_apply_peripheral_configuration() call due to A_HW_CONFIG_ERROR, the application can use this API to get the reason if the error is due to pin conflict between peripherals.	
Prototype	A_STATUS qcom_gpio_peripheral_pin_conflict_check(
	A_UINT32 peripheral_id,	Peripheral ID for which the failure check is performed.
	A_UINT32 *pPeripheralMap	An OUT pointer filled by this API. If peripheral is successfully enabled, this point is set to 0. If failed, the API returns nonzero value containing the bitmap of peripheral IDs which caused the failure.
)	
Return value	A_OK: The requested peripheral information is successfully obtained. A_HW_CONFIG_ERROR: Requested peripheral information cannot be obtained. One of the possible reason is that the active configuration is not provided for the requested peripheral.	

7.8.4 qcom_gpio_pin_dir

Definition	Set the direction for the given hardware pin when it operates in GPIO mode.	
Prototype	A_STATUS qcom_gpio_pin_dir(
	A_UINT32 pin,	Hardware pin number
	A_BOOL input	A_TRUE = Configure the pin as input. A_FLASE = Configure the pin as output.
)	
Return value	A_OK on success, A_ERROR on failure (for example, the given hardware pin is not in GPIO mode)	

7.8.5 qcom_gpio_pin_set

Definition	Set the output value for the given hardware pin when it operates in GPIO mode. This API is used only when the given pin's direction is set to output by calling qcom_gpio_pin_dir().	
Prototype	A_STATUS qcom_gpio_pin_set(
	A_UINT32 pin,	Hardware pin number
	A_BOOL input	A_TRUE = Drive logic high. A_FLASE = Drive logic low.
)	
Return value	A_OK on success, A_ERROR on failure (for example, the given hardware pin is not in GPIO mode)	

7.8.6 qcom_gpio_get_interrupt_pin_num

Definition	This API obtains the hardware GPIO pin number for the given functional interrupt pin. List of functional interrupts are provided in <code>qcom_gpio_interrupts.h</code> . Applications should obtain the hardware pin number using this API before calling any other APIs that takes hardware pin number as the parameter (like interrupt registration/deregistration/GPIO pin configuration update (pin_pad, direction, source, and so forth.)).	
Prototype	<code>A_STATUS</code> <code>qcom_gpio_get_interrupt_pin_num(</code>	
	<code> A_UINT32 interrupt_num,</code>	Software interrupt number defined by GPIO framework in <code>qcom_gpio_interrupts.h</code> for which the hardware pin numbers must be obtained.
	<code> A_UINT32 *pPinNum</code>	An INOUT pointer. This is an address of the variable in which the hardware pin number is filled. This parameter is filled with <code>QCOM_GPIO_HW_PIN_NOT_SET</code> in case of failure.
	<code>)</code>	
Return value	<code>A_OK</code> on success, <code>A_ERROR</code> on failure (for example, interrupt number belongs to a disabled peripheral).	

7.8.7 qcom_gpio_interrupt_register

Definition	Register an interrupt handler for the given hardware pin. The pin number defined in the <code>qcom_gpio_interrupt_info_t</code> specifies the hardware pin number and the interrupt handler function pointer. Since the hardware pin number is not exported to application, application needs to obtain the hardware pin number by calling <code>qcom_gpio_get_interrupt_pin_num()</code> and pass it on to this API.	
Prototype	<code>A_STATUS qcom_gpio_interrupt_register(</code>	
	<code> qcom_gpio_interrupt_info_t *gpio_interrupt_info</code>	<p>This parameter contains the information such as hardware pin number, function pointer, and flags for interrupt handler. The structure looks like the following:</p> <pre>typedef struct _qcom_gpio_interrupt_info_t { int pin; void (*gpio_pin_int_handler_fn)(void *arg); void *arg; unsigned int internal_flags; } qcom_gpio_interrupt_info_t;</pre> <p><code>internal_flags</code> should contain the interrupt mode. The interrupt mode can take one of the following values.</p> <pre>typedef enum { QCOM_GPIO_PIN_INT_NONE = 0, QCOM_GPIO_PIN_INT_RISING_EDGE, QCOM_GPIO_PIN_INT_FALLING_EDGE, QCOM_GPIO_PIN_INT_BOTH_EDGE, QCOM_GPIO_PIN_INT_LEVEL_LOW, QCOM_GPIO_PIN_INT_LEVEL_HIGH, } QCOM_GPIO_PIN_INTERRUPT_MODE;</pre>
	<code>)</code>	
Return value	<code>A_OK</code> on success, <code>A_ERROR</code> on failure (for example, invalid hardware pin, internal memory allocation failure).	

7.8.8 qcom_gpio_interrupt_mode

Definition	<p>Configure the interrupt trigger mode (level 0/1 or rising/falling/any edge) for the given hardware pin.</p> <p>This API is used only when the given hardware pin is configured to GPIO mode. When a pin is configured in peripheral mode, the interrupt mode is expected to change in the runtime and is provided as part of the tunable active configuration.</p> <p>The pin number defined in the <code>qcom_gpio_interrupt_info_t</code> specifies the hardware pin number and the interrupt handler function pointer. Since the hardware pin number is not exported to application, application needs to obtain the hardware pin number by calling <code>qcom_gpio_get_interrupt_pin_num()</code> and pass it on to this API.</p>	
Prototype	A_STATUS <code>qcom_gpio_interrupt_mode(</code>	
	<code>qcom_gpio_interrupt_info_t</code> <code>*gpio_interrupt_info,</code>	<p>This parameter contains the information such as hardware pin number, function pointer, and flags for interrupt handler. The structure looks like the following:</p> <pre>typedef struct _qcom_gpio_interrupt_info_t { int pin; void (*gpio_pin_int_handler_fn)(void *arg); void *arg; unsigned int internal_flags; } qcom_gpio_interrupt_info_t;</pre> <p>Do not modify the <code>internal_flags</code> parameter. It is used by GPIO framework internally.</p> <p>The <code>gpio_pin_int_handler_fn</code> parameter can be set to NULL for this API.</p>
	<code>QCOM_GPIO_PIN_INTERRUPT_MODE</code> <code>mode</code>	<p>Specifies the interrupt mode to be configured for the given pin specified in <code>gpio_interrupt_info</code>. The mode can take one of the following enumerations:</p> <pre>typedef enum { QCOM_GPIO_PIN_INT_NONE = 0, QCOM_GPIO_PIN_INT_RISING_EDGE, QCOM_GPIO_PIN_INT_FALLING_EDGE, QCOM_GPIO_PIN_INT_BOTH_EDGE, QCOM_GPIO_PIN_INT_LEVEL_LOW, QCOM_GPIO_PIN_INT_LEVEL_HIGH, } QCOM_GPIO_PIN_INTERRUPT_MODE;</pre>
	<code>)</code>	
Return value	A_OK on success, A_ERROR on failure (for example the given hardware pin is not in GPIO mode)	

NOTE: Changes to interrupt mode using this API are not saved and restored across suspend/resume. This will be fixed in the next release.

7.8.9 qcom_gpio_interrupt_wakeup

Definition	Set the wakeup interrupt for the given hardware pin. This API is used only when the given hardware pin is configured to GPIO mode.	
Prototype	A_STATUS <code>qcom_gpio_interrupt_wakeup(</code>	

	<i>A_UINT32 pin,</i>	Hardware pin
	<i>A_BOOL enable</i>	A_TRUE = Set the pin as wakeup interrupt. A_FALSE = The pin is not a wakeup interrupt.
)	
Return value	A_OK on success, A_ERROR on failure (for example the given hardware pin is not in GPIO mode).	

7.8.10 qcom_gpio_pin_source

Definition	Set the source of the given hardware pin. This API is used only when the given hardware pin is configured to GPIO mode.	
Prototype	A_STATUS qcom_gpio_pin_source(
	<i>A_UINT32 pin,</i>	Hardware pin
	<i>A_BOOL pwm</i>	A_TRUE = Sigma-Delta PWM source A_FALSE = WLAN_GPIO_OUT register
)	
Return value	A_OK on success, A_ERROR on failure (for example, the given hardware pin is not in GPIO mode).	

7.8.11 qcom_i2cm_init

Definition	Initialize I ² C master bus.	
Prototype	A_STATUS qcom_i2cm_init (
	<i>I2C_PIN_PAIR pin_pair_id</i>	I ² C SDA/SCK GPIO pin pair ID. Range: 0-4
	<i>I2C_FREQ freq</i>	I ² C clock frequency. Range: 0-9
	<i>A_UINT32 timeout</i>	Clock stretch timeout. Range: 0-1285000 μ s depending on frequency. 0 for default value: 35000 μ s
)	
Return value	0 on success or error code on failure	

7.8.12 qcom_i2cm_read

Definition	This function is used to I ² C master read control.	
Prototype	A_STATUS qcom_i2cm_read (
	<i>I2C_PIN_PAIR pin_pair_id</i>	I ² C SDA/SCK GPIO pin pair ID. Range: 0-4
	<i>A_UINT8 dev_addr</i>	I ² C device ID, unique 7-bit address
	<i>A_UINT32 word_addr</i>	word/reg address of I ² C device to read. Ignore it in simple read format
	<i>A_UINT8 addr_len</i>	Length of word_addr data, 1-6 byte is preferred to combined format, 0 for simple read format
	<i>A_UCHAR *data</i>	The buffer pointer to the Rx data

	<i>A_UINT32 data_len</i>	Length of Rx chunk data, 0-8 byte is preferred
)	
Return value	0 on success, -1 on failure, -2 on timeout	

7.8.13 qcom_i2cm_write

Definition	This function is used to I ² C write control.	
Prototype	<i>A_INT32 qcom_i2cm_write (</i>	
	<i>I2C_PIN_PAIR pin_pair_id</i>	I ² C SDA/SCK GPIO pin pair ID. Range: 0-4
	<i>A_UINT8 dev_addr</i>	I ² C device ID, unique 7-bit address
	<i>A_UINT32 word_addr</i>	word/reg address of I ² C device to read. Ignore it in simple write format
	<i>A_UINT8 addr_len</i>	Length of word_addr data, 1-7 byte is preferred to combined format, 0 for simple write format
	<i>A_UCHAR *data</i>	The buffer pointer to the data to write
	<i>A_UINT32 data_len</i>	Length of write data, 1-8 byte is preferred
)	data_len+addr_len should not be larger than 7
Return value	0 on success, -1 on failure, -2 on timeout	

7.8.14 qcom_i2cm_fini

Definition	This function is used to de-initialize I ² C module in I ² C fd.	
Prototype	<i>A_INT32 qcom_i2cm_fini (</i>	
	<i>I2C_PIN_PAIR pin_pair_id</i>	I ² C SDA/SCK GPIO pin pair ID
)	
Return value	0 on success or error code on failure	
Implantation	Restore setting, disable module NOTE: This API does not work before calling corresponding qcom_i2c_init.	

7.8.15 qcom_i2cs_control

Definition	This function is used to initialize I ² C slave module.	
Prototype	<i>A_INT32 qcom_i2cs_control (</i>	
	<i>int enable</i>	Enable/disable module
	<i>I2CS_PIN_PAIR pin_pair_id</i>	I ² C slave SDA/SCK GPIO pin pair ID. Range: 0-1
)	
Return value	0 on success or error code on failure	

7.8.16 qcom_i2cs_cmd

Definition	This function is used to send command to I ² C slave module	
Prototype	A_INT32 qcom_i2cs_cmd (
	A_UINT32 cmd	I ² C slave command types
	A_UINT32 *data	I ² C slave data used for both user to service and service to user info exchange
)	
Return value	0 on success or error code on failure	

7.8.17 qcom_i2cs_csr_init

Definition	This function is used to initialize I ² C slave CSR module (single byte read/write)	
Prototype	A_INT32 qcom_i2cs_csr_init(
	I2CS_CSR_PARA *csr_params	CSR service registration parameters
)	
Return value	0 on success or error code on failure	

7.8.18 qcom_i2cs_reg_init

Definition	This function is used to initialize I ² C slave register module (4 byte read/write)	
Prototype	A_INT32 qcom_i2cs_reg_init(
	I2CS_REG_PARA *reg_params	Register service registration parameters
)	
Return value	0 on success or error code on failure	

7.8.19 qcom_i2cs_fifo_init

Definition	This function is used to initialize I ² C slave FIFO module (page read/write)	
Prototype	A_INT32 qcom_i2cs_fifo_init(
	I2CS_FIFO_PARA *fifo_params	FIFO service registration parameters
)	
Return value	0 on success or error code on failure	

7.8.20 qcom_pwm_control

Description	Start/stop one set of PWM function	
Prototype	A_STATUS qcom_pwm_control(
	A_BOOL module_select,	0 = PWM port module 1 = Sigma delta module

	<i>A_BOOL enable</i>	TRUE: enable PWM output in port_id channel in module_select mode FALSE: disable PWM output in port_id channel in module_select mode
	<i>A_UINT32 port_group,</i>	Indicate which channel of PWM to be configured. Set it based on MACRO PWM_PORT_n
)	
Return value	A_OK / A_ERROR	
Implantation	Enable/disable the SDM/PWM PORT of port_id NOTE: Sigma delta module is not supported in sleep mode	

7.8.21 qcom_pwm_sdm_set

Description	Configure frequency, duty cycle and dimming level of a given set of PWM function	
Prototype	<code>void qcom_pwm_set(</code>	
	<i>A_UINT32 freq,</i>	Configure the frequency, unit: KHz
	<i>A_UINT32 duty,</i>	Range: 0-255; for duty cycle of percentage format, divided by 256.
	<i>A_UINT32 pwm_id,</i>	Indicate which channel of PWM to be configured. It should not be larger than the MACRO PWM_MAX_CHANNEL.
)	
Return value	A_OK / A_ERROR	
Implantation	Set and calculate parameters NOTE: If a very stable and reliable driving waveform is required, call qcom_pwm_sdm_set before enabling the port (call qcom_pwm_port_control(1,1,port_id)) when initializing a port, so the PWM wave will not generate a spur at the beginning.	

7.8.22 qcom_pwm_port_set

Description	Configure frequency, duty cycle and dimming level of a given set of PWM function	
Prototype	<code>void qcom_pwm_port_set(</code>	
	<i>A_UINT32 freq,</i>	Configure the frequency, has a maximum precision of 0.01Hz, 10000 equal to 100.00Hz
	<i>A_UINT32 duty,</i>	Range: 0-10000; for duty cycle of percentage format, has a maximum precision of 0.01% ,10000 means 100.00%
	<i>A_UINT32 phase,</i>	Range: 0-10000; for phase shift of percentage format, has a maximum precision of 0.01% ,10000 equal to 100.00%
	<i>A_UINT32 pwm_id,</i>	Indicate which channel of PWM to be configured. Should not be larger than the MACRO PWM_MAX_CHANNEL

	<i>A_UINT32 src_clk</i>	Range: 0-2. Source clock. 0 = CPU_CLK 1 = REF_CLK 2 = LPO_CLK
)	
Return value	A_OK / A_ERROR	
Implantation	Set parameters NOTE: CPU_CLK & REF_CLK are not available in deep sleep mode. If set, clock will stop. All eight PWM channels share the same source clock, changing src_clk will effect setting of ALL available channels. 1. Disable all port currently used 2. Change source clock (call qcom_pwm_port_set) (If the source clock will change to LPO_CLK or from LPO_CLK, repeat step 2 to call qcom_pwm_set in all channels currently used.) 3. Enable all port used. If a very stable and reliable driving waveform is required, call qcom_pwm_port_set before enabling the port (that is, calling qcom_pwm_port_control) when initializing a port, so the PWM wave will not generate a spur at the beginning.	

7.8.23 qcom_spi_init

Definition	Initiate legacy SPI interface.	
Prototype	int qcom_spi_init (
	<i>SPI_CS_PIN cs_pin</i>	SPI chip select pin
	<i>SPI_MOSI_PIN mosi_pin</i>	SPI MOSI pin
	<i>SPI_MISO_PIN miso_pin</i>	SPI MISO pin
	<i>SPI_SCLK_PIN sclk_pin</i>	SPI SCLK pin
	<i>SPI_FREQ freq</i>	SPI frequency
)	
Return value	-1/0	

7.8.24 qcom_spi_fini

Definition	Restore GPIO pins used by SPI and disable SPI function.	
Prototype	void qcom_spi_fini (
)	
Return value	void	

7.8.25 qcom_spi_wait_done

Definition	Wait SPI session to finish.	
Prototype	A_UINT32 qcom_spi_wait_done (
)	
Return value	0 for success, nonzero for failure	

7.8.26 qcom_spi_request

Definition	Initiate SPI session.	
Prototype	void qcom_spi_request (
	A_UINT32 *req	Buffer pointer to be transferred
	unsigned int rxcnt	The number of bytes (from 0 to 8) to receive on the SPI
	unsigned int txcnt	The number of bytes (from 0 to 8) to send on the SPI
)	
Return value	void	

7.8.27 qcom_spi_response

Definition	Read from the attached SPI device.	
Prototype	void qcom_spi_response (
	A_UINT8 *buf	Buffer used for receiving data
	unsigned int nbytes	The number of bytes (up to 8)
)	
Return value	void	

7.8.28 qcom_i2s_init

Definition	Initialize I ² S function. This API can be used to: <ul style="list-style-type: none"> Initialize I²S interface 0/1, and select the frequency. Call the callback function in the descriptor when the receive interrupt is triggered. Use qcom_i2s_rcv_ctrl to start and stop the receive DMA. 	
Prototype	A_INT32 qcom_i2s_init (
	typedef struct {	Port: 0 = I ² S0, 1 = I ² S1
	A_UINT32 port;	
	I2S_FREQ freq;	Freq: typedef enum {
	A_UINT8 dsize;	I2S_FREQ_44_1K = 0,
	A_UINT32 i2s_buf_size;	I2S_FREQ_48K = 1,
	A_UINT32 num_rx_desc;	I2S_FREQ_32K = 2,
	A_UINT32 num_tx_desc;	I2S_FREQ_96K = 3
	A_UINT8 i2s_mode;	}I2S_FREQ;
	} i2s_app_config;	Dsize: The word size of each channel (8, 16, 24, 32 bits).
		Tx buffer count in DMA. Buffer size over 4 is recommended.
		Rx buffer count in DMA. Buffer size over 4 is recommended.
		Buffer size. 1024 is recommended.
		i2s_mode: 1 = master mode; 0 = slave mode.

	<code>void *intr_cb_txcomp</code>	<code>void intr_cb_txcomp (void *)</code> This callback is used when I ² S DMA transmit is completed. The user can continue to write DMA when this interrupt is received, otherwise it means DMA is not finished. If this parameter is NULL, the user will use the polling method to transmit the data.
	<code>void *intr_cb_rx</code>	<code>void intr_cb_rx (void *)</code> This callback is used in ISR mode. For example, it calls WMI interface to release a signal to notify or wake some thread. The user then calls <code>qcom_i2s_rcv_data(...)</code> to receive the data. If this parameter is NULL, the user will use the polling method to receive the data.
	<code>)</code>	
Return value	A_OK on success, A_ERROR on error	

7.8.29 qcom_i2s_rcv_control

Definition	Start processing data from an I ² S interface. This API is used to start or stop the DMA channel to get data from the I ² S interface.	
Prototype	<code>void qcom_i2s_rcv_control (</code>	
	<code>A_UINT8 port,</code>	0 = I ² S0, 1 = I ² S1
	<code>I2S_REV_CONTROL control</code>	<code>typedef enum {</code> <code> I2S_RCV_START = 0,</code> <code> I2S_RCV_STOP = 1,</code> <code>}I2S_RCV_CONTROL;</code>
	<code>)</code>	
Return value	N/A	

7.8.30 qcom_i2s_rcv_data

Definition	Receive data from an I ² S interface. This API gets the data in I ² S functionality from high level.	
Prototype	<code>A_UINT32 qcom_i2s_rcv_data (</code>	
	<code>A_UINT8 port,</code>	0 = I ² S0, 1 = I ² S1
	<code>A_UINT8 *data,</code>	Data buffered from the I ² S interface
	<code>A_UINT32 bufLen,</code>	Maximum data buffer length. Usually 1k.
	<code>A_UINT32 *len</code>	Valid data length returned
	<code>)</code>	
Return value	1 = Data remaining to be received 0 = No data remaining to be received	

7.8.31 qcom_i2s_xmt_data

Definition	Transmit data to an I ² S interface. This API sends the data to I ² S interface from high level.	
Prototype	A_UINT32 qcom_i2s_xmt_data (
	A_UINT8 port,	0 = I ² S0, 1 = I ² S1
	A_UINT8 *data,	Data buffered to be transmitted to the I ² S interface
	A_UINT32 *len	Data length
)	
Return value	Greater than 0 = Data remaining to be processed 0 = No data remaining to be processed	

7.8.32 qcom_adc_init

Description	Open the ADC according to the specified parameters	
Prototype	int qcom_adc_init(
	A_UINT8 gain_scale,	0: 1.8V, 1: 3.3V
	A_UINT8 accuracy,	Sample accuracy, 6~12
	A_UINT8 freq,	Sample frequency
	A_UINT8 input_type,	0: single ended input 1: differential
	A_BOOL stream_bit	Default is 0, set to 1 when using voice stream
)	
Return value	0 for success, nonzero for failure	

7.8.33 qcom_adc_config

Description	Configure the ADC	
Prototype	int qcom_adc_config (
	A_UINT8 scan_mode,	Scan Mode Enable 0: one-channel working 1: multi-channel working
	A_UINT8 trigger_mode,	Conversion Trigger Select 0: software trigger 1: hardware timer trigger 2: hardware pin trigger
	A_UINT8 adco_mode,	ADC work mode 0: sample-by-sample(single)(static scenario) 1: continuous(dynamic scenario), if SCAN=1, repeat scan
	ADC_CHAN_CFG_T channel[]	Channel configuration
	A_UINT8 channel_cnt	Channel number
)	
Return value	0 for success, nonzero for failure	

7.8.34 qcom_adc_timer_config

Description	Configure the ADC timer when using timer trigger mode	
Prototype	int qcom_adc_timer_config (
	<i>A_UINT32 time_start,</i>	Timer start value (default is 300 ms) Sets the timer start value. The timer counts down until it reaches 0, and then loads this register value again and counts down.
	<i>A_UINT32 time_unit</i>	Timer start value unit 0: unit is clock period (CLK=66M) 1: 1 μ s 2: 1 ms 3: 10 ms
)	
Return value	0 for success, nonzero for failure	

7.8.35 qcom_adc_dma_config

Description	Configure the ADC DMA	
Prototype	int qcom_adc_dma_config (
	<i>A_UINT32 threshold,</i>	Threshold for channel data cnt
	<i>A_UINT32 data_cnt</i>	Total data cnt of all channels for DMA. When exhaust will disable DMA input and transfer. When set to 0, no limit for transfer data byte number.
)	
Return value	0 for success, nonzero for failure	

7.8.36 qcom_adc_calibration

Description	Do the ADC calibration process	
Prototype	int qcom_adc_calibration (
)	
Return value	0 for success, nonzero for failure	

7.8.37 qcom_adc_conversion

Description	Start/stop adc conversion	
Prototype	int qcom_adc_conversion (
	<i>A_BOOL start</i>	1: start conversion 0: stop conversion
)	
Return value	0 for success, nonzero for failure	

7.8.38 qcom_adc_recv_data

Description	Get ADC conversion results	
Prototype	int qcom_adc_read(
	<i>A_UINT8 channel,</i>	ADC channel, 1-8
	<i>A_UINT8* buffer,</i>	Data buffer
	<i>A_UINT32 buf_len,</i>	Buffer length
	<i>A_UINT32* ret_len,</i>	Return length
	<i>A_UINT8* more,</i>	1: there is more data, 0: there is no data
	<i>A_UINT8* done</i>	1: ADC conversion done
)	
Return value	0 for success, nonzero for failure	

7.8.39 qcom_adc_close

Description	Close ADC, set ADC to default settings	
Prototype	int qcom_adc_close(
)	
Return value	0 for success, nonzero for failure	

7.8.40 qcom_wkup_pin_config

Definition	Set wakeup pin status.	
Prototype	A_UNIT8 qcom_wkup_pin_config (
	<i>A_UNIT8 gpio</i>	WKUP_PIN_30 WKUP_PIN_31
	<i>A_UINT8 dir</i>	WKUP_PIN_OUT WKUP_PIN_IN
	<i>A_UINT8 pull</i>	WKUP_PIN_PULL_NONE WKUP_PIN_PULL_DOWN WKUP_PIN_PULL_UP
)	
Return value	void	

7.8.41 qcom_uart_rx_pin_set

Definition	Set UART Rx pin.	
Prototype	void qcom_uart_rx_pin_set (
	<i>int pin0,</i>	Rx pin for UART0. Default: 2
	<i>int pin1</i>	Rx pin for UART1. Default: 10
)	
Return value	void	

7.8.42 qcom_uart_tx_pin_set

Definition	Set UART Tx pin.	
Prototype	void qcom_uart_tx_pin_set (
	<i>int pin0</i> ,	Tx pin for UART0. Default: 7.
	<i>int pin1</i>	Tx pin for UART1. Default: 11.
)	
Return value	void	

7.8.43 qcom_dset_create

Definition	Create data set on indicated media.	
Prototype	A_STATUS qcom_dset_create (
	DSET_HANDLE *dset_handle,	Storage address for return data set handle. Default = 0.
	DSET_ID dset_id,	Data set ID
	uint32_t length,	Data set length
	uint32_t flags,	Media ID on which the data set is to be created. Default = 0 Media IDs: DSET_MEDIA_HOST 1 //host mode only DSET_MEDIA_RAM 2 DSET_MEDIA_NVRAM 3 DSET_MEDIA_OTP 4 //very small, one-time programming DSET_MEDIA_ROM 5 DSET_MEDIA_MEM 6 // The combination of (DSET_MEDIA_RAM + DSET_MEDIA_ROM) DSET_MEDIA_AORAM 7
	DSET_CB create_cb,	Asynchronous callback function. May be NULL.
	void *cb_arg	Callback function parameter. May be NULL.
)	
Return value	A_OK, A_FAIL, A_PENDING	

7.8.44 qcom_dset_open

Definition	Open data set for access.	
Prototype	A_STATUS qcom_dset_open (
	DSET_HANDLE *dset_handle,	Storage address for return data set handle. Default = 0.
	DSET_ID dset_id,	Data set ID

	<i>uint32_t flags,</i>	If the flag is greater or less than 0, this is the start media ID the open operation will be performed on. Otherwise the open operation will traverse across all supported media. Media ID and traverse order: DSET_MEDIA_HOST 1 //host mode only DSET_MEDIA_RAM 2 DSET_MEDIA_NVRAM 3 DSET_MEDIA_OTP 4 //very small, one-time programming DSET_MEDIA_ROM 5 DSET_MEDIA_MEM 6 // The combination of (DSET_MEDIA_RAM + DSET_MEDIA_ROM) DSET_MEDIA_AORAM 7
	<i>DSET_CB open_cb,</i>	Asynchronous callback function. May be NULL.
	<i>void *cb_arg</i>	Callback function parameter. May be NULL.
)	
Return value	A_OK, A_FAIL, A_PENDING	

7.8.45 qcom_dset_read

Definition	Read data from data set to buffer.	
Prototype	A_STATUS qcom_dset_read (
	<i>DSET_HANDLE dset_handle,</i>	Data set handle
	<i>uint8_t *buffer,</i>	Data buffer
	<i>uint32_t length,</i>	Data length to read
	<i>uint32_t offset,</i>	Offset in data set to start reading. Default = 0
	<i>DSET_CB read_cb,</i>	Asynchronous callback function. May be NULL.
	<i>void *cb_arg</i>	Callback function parameter. May be NULL.
)	
Return value	A_OK = Read operation complete. Data in buffer ready for use. A_FAIL = Read operation failed. Data in buffer invalid. A_PENDING = Asynchronous operation. Data in buffer is not ready for use.	

7.8.46 qcom_dset_write

Definition	Write data to data set.	
Prototype	A_STATUS qcom_dset_write (
	<i>DSET_HANDLE dset_handle,</i>	Data set handle
	<i>uint8_t *buffer,</i>	Data buffer
	<i>uint32_t length,</i>	Data length to write
	<i>uint32_t offset,</i>	Offset in data set to start writing. Default = 0
	<i>uint32_t flags,</i>	Only support circular operation. Default = 0
	<i>DSET_CB write_cb,</i>	Asynchronous callback function. May be NULL.
	<i>void *cb_arg</i>	Callback function parameter. May be NULL.
)	

Return value	A_OK, A_FAIL, A_PENDING
---------------------	-------------------------

7.8.47 qcom_dset_close

Definition	Close the opened data set and release resource.	
Prototype	A_STATUS qcom_dset_close (
	<i>DSET_HANDLE</i> <i>dset_handle</i> ,	Data set handle
	<i>DSET_CB</i> <i>close_cb</i> ,	Asynchronous callback function. May be NULL.
	<i>void</i> * <i>cb_arg</i>	Callback function parameter. May be NULL.
)	
Return value	A_OK, A_FAIL, A_PENDING	

7.8.48 qcom_dset_commit

Definition	Commit cached data after data set is created on media.	
Prototype	A_STATUS qcom_dset_commit (
	<i>DSET_HANDLE</i> <i>dset_handle</i> ,	Data set handle
	<i>DSET_CB</i> <i>commit_cb</i> ,	Asynchronous callback function. May be NULL.
	<i>void</i> * <i>cb_arg</i>	Callback function parameter. May be NULL.
)	
Return value	A_OK, A_FAIL, A_PENDING	

7.8.49 qcom_dset_delete

Definition	Delete data set from media.	
Prototype	A_STATUS qcom_dset_delete (
	<i>DSET_ID</i> <i>dset_id</i> ,	Data set ID
	<i>A_UINT32</i> <i>flags</i> ,	The media ID that the data set resides on. Media ID and traverse order: DSET_MEDIA_HOST 1 //host mode only DSET_MEDIA_RAM 2 DSET_MEDIA_NVRAM 3 DSET_MEDIA_OTP 4 //very small, one-time programming DSET_MEDIA_ROM 5 DSET_MEDIA_MEM 6 // The combination of (DSET_MEDIA_RAM+ DSET_MEDIA_ROM) DSET_MEDIA_AORAM 7
	<i>DSET_CB</i> <i>delete_cb</i> ,	Asynchronous callback function. May be NULL.
	<i>void</i> * <i>cb_arg</i>	Callback function parameter. May be NULL.
)	
Return value	A_OK, A_FAIL, A_PENDING	

7.8.50 qcom_dset_size

Definition	Return data set size.	
Prototype	A_UINT32 qcom_dset_size (
	DSET_HANDLE dset_handle,	Data set handle
)	
Return value	Data set size	

7.8.51 qcom_dset_media

Definition	Get media ID that the data set resides on.	
Prototype	A_UINT32 qcom_dset_media (
	DSET_HANDLE dset_handle,	Data set handle
)	
Return value	Data set media ID.	

7.9 Unified Security APIs

7.9.1 Key management APIs

The QCA4010/QCA4012 SDK exposes a set of APIs to application developers to manage keys that are used in cryptographic operations.

QCA4010/QCA4012 does not provide a separate Trusted Execution Environment (TEE) and therefore all applications running on the QCA4010/QCA4012 platform are assumed to be trusted applications.

Crypto operations (with the exception of digest) require an object handle. The object handle contains the underlying raw key material.

- Applications allocate transient objects that are held in memory and are automatically wiped and reclaimed when they are freed.
 - Transient objects are used for cryptographic keys and key-pairs.
 - Transient objects contain only attributes and no data stream.
 - A transient object can be uninitialized, in which case it is an object container allocated with a certain object type and maximum size but with no attributes. A transient object becomes initialized when its attributes are populated.
 - Transient objects are manipulated through object handles.
- To store persistent objects, use qcom_dset* QAPIs. See section [7.8.43](#) to section [7.8.51](#) for more details.

7.9.1.1 Unified Security data types

This section describes various data structures exported by the Qualcomm Technologies library for security/cryptographic usage.

qcom_crypto_attr_t

An array of this type is passed whenever a set of attributes is specified as argument to function of the API. An attribute can be either a buffer attribute or a value attribute. This is determined by bit [29] of the attribute identifier. If this bit is set to 0, the attribute is a buffer attribute and the field **ref** MUST be selected. If the bit is set to 1, it is a value attribute and the field **value** MUST be selected.

```
typedef struct {
    A_UINT32 attribute_id;
    union
    {
        struct
        {
            [inbuf] void* buffer; A_UINT32 length;
        } ref;
        struct
        {
            A_UINT32 a, b;
        } value;
    } content;
} qcom_crypto_attr_t;
```

qcom_crypto_obj_info_t

```
typedef struct {
    A_UINT32 objectType;
    A_UINT32 keySize;
    A_UINT32 maxKeySize;
    A_UINT32 objectUsage;
    A_UINT32 handleFlags;
} qcom_crypto_obj_info_t;
```

qcom_crypto_obj_hdl_t

qcom_crypto_obj_hdl_t is an opaque handle on a cryptographic object. These handles are returned by the function **qcom_crypto_transient_obj_alloc**.

```
typedef A_UINT32 qcom_crypto_obj_hdl_t;
```

Table 7-1 Usage constants

Constant name	Value
QCOM_CRYPT0_USAGE_EXTRACTABLE	0x00000001
QCOM_CRYPT0_USAGE_ENCRYPT	0x00000002
QCOM_CRYPT0_USAGE_DECRYPT	0x00000004
QCOM_CRYPT0_USAGE_MAC	0x00000008
QCOM_CRYPT0_USAGE_SIGN	0x00000010
QCOM_CRYPT0_USAGE_VERIFY	0x00000020
QCOM_CRYPT0_USAGE_DERIVE	0x00000040

Table 7-2 Handle flag constants

Constant name	Value
Unused	0x00010000
QCOM_CRYPT0_HANDLE_FLAG_INITIALIZED	0x00020000
QCOM_CRYPT0_HANDLE_FLAG_KEY_SET	0x00040000
Unused	0x00080000

Table 7-3 Operation constants

Constant Name	Value
QCOM_CRYPT0_OPERATION_CIPHER	1
QCOM_CRYPT0_OPERATION_MAC	3
QCOM_CRYPT0_OPERATION_AE	4
QCOM_CRYPT0_OPERATION_DIGEST	5
QCOM_CRYPT0_OPERATION_ASYMMETRIC_CIPHER	6
QCOM_CRYPT0_OPERATION_ASYMMETRIC_SIGNATURE	7
QCOM_CRYPT0_OPERATION_KEY_DERIVATION	8
Reserved for future use	0x00000009-0x7FFFFFFF
Implementation defined	0x80000000-0xFFFFFFFF

Table 7-4 Operation states

Constant Name	Value
QCOM_CRYPT0_OPERATION_STATE_INITIAL	0x00000000
QCOM_CRYPT0_OPERATION_STATE_ACTIVE	0x00000001
Reserved for future use	0x00000002-0x7FFFFFFF
Implementation defined	0x80000000-0xFFFFFFFF

7.9.1.2 qcom_crypto_obj_info_get

Definition	Return the characteristics of an object.	
Prototype	A_CRYPT0_STATUS qcom_crypto_obj_info_get (
	<i>qcom_crypto_obj_hdl_t object</i>	Handle of the object
	<i>qcom_crypto_obj_info_t* obj_info</i>	Pointer to a structure filled with the object information
)	

Description	<p>It fills in the following fields in the structure <code>qcom_crypto_object_info_t</code> (section 7.9.1.1):</p> <ul style="list-style-type: none"> ▪ <code>object_type</code>: The parameter <code>object_type</code> passed when the object was created. ▪ <code>key_size</code>: The current size in bits of the object as determined by its attributes. This will always be less than or equal to <code>max_key_size</code>. Set to 0 for uninitialized and data only objects. ▪ <code>max_key_size</code>: The maximum <code>key_size</code> which this object can represent. This is set to the parameter <code>max_key_size</code> passed to <code>qcom_crypto_transient_obj_alloc</code> ▪ <code>obj_usage</code>: A bit vector of the <code>QCOM_CRYPT0_USAGE_XXX</code> bits defined in Table 7-1. ▪ <code>handle_flags</code>: A bit vector containing one or more of the following flags: <ul style="list-style-type: none"> ▫ <code>QCOM_CRYPT0_HANDLE_FLAG_INITIALIZED</code> <ul style="list-style-type: none"> – For a transient object, initially cleared, then set when the object becomes initialized.
Return value	<code>A_CRYPTO_SUCESS</code> / <code>A_CRYPTO_ERROR</code>

7.9.1.3 `qcom_crypto_obj_usage_restrict`

Definition	Restrict the object usage flags of an object handle to contain at most the flags passed in the <code>obj_usage</code> parameter.	
Prototype	<code>A_CRYPTO_STATUS</code>	
	<code>qcom_crypto_obj_usage_restrict(</code>	
	<code>qcom_crypto_obj_hdl_t object,</code>	Handle on an object
	<code>uint32_t obj_usage</code>	New object usage, an OR combination of one or more of the <code>QCOM_CRYPT0_USAGE_XXX</code> constants defined in Table 7-1
	<code>)</code>	
Description	<p>For each bit in the parameter <code>obj_usage</code>:</p> <ul style="list-style-type: none"> ▪ If the bit is set to 1, the corresponding usage flag in the object is left unchanged. ▪ If the bit is set to 0, the corresponding usage flag in the object is cleared. <p>For example, if the usage flags of the object are set to <code>QCOM_CRYPT0_USAGE_ENCRYPT QCOM_CRYPT0_USAGE_DECRYPT</code> and if <code>obj_usage</code> is set to <code>QCOM_CRYPT0_USAGE_ENCRYPT QCOM_CRYPT0_USAGE_EXTRACTABLE</code>, then the only remaining usage flag in the object after calling the function <code>qcom_crypto_obj_usage_restrict</code> is <code>QCOM_CRYPT0_USAGE_ENCRYPT</code>.</p> <p>Note that an object usage flag can only be cleared. A transient object's usage flags are reset to 1 using the <code>qcom_crypto_transient_obj_reset</code> function.</p>	
Return value	<code>A_CRYPTO_SUCESS</code> / <code>A_CRYPTO_ERROR</code>	

7.9.1.4 `qcom_crypto_obj_buf_attr_get`

Definition	Extract one buffer attribute from an object.	
Prototype	<code>A_CRYPTO_STATUS</code>	
	<code>qcom_crypto_obj_buf_attr_get(</code>	
	<code>qcom_crypto_obj_hdl_t object,</code>	Handle of the object
	<code>A_UINT32 attribute_id,</code>	Identifier of the attribute to retrieve
	<code>void* buffer, A_UINT32* size</code>	Output buffer to get the content of the attribute

)	
Description	<p>The attribute is identified by the argument <code>attribute_id</code>. The precise meaning of this parameter depends on the container type and size and is defined in section 7.9.3.4. Bit [29] of the attribute identifier MUST be set to 0, that is, it MUST denote a buffer attribute.</p> <p>They are two kinds of object attributes, which are identified by a bit in their handle value (see Table 7-19):</p> <ul style="list-style-type: none"> Public object attributes can always be extracted whatever the status of the container. Protected attributes can be extracted only if the object's key usage contains the <code>QCOM_CRYPT_USAGE_EXTRACTABLE</code> flag. <p>See section 7.9.3.4 for a definition of all available object attributes, their formats, and their level of protection.</p>	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.1.5 qcom_crypto_obj_val_attr_get

Definition	Extract a value attribute from an object.	
Prototype	A_CRYPTO_STATUS	
	qcom_crypto_obj_val_attr_get(
	<i>qcom_crypto_obj_hdl_t</i> <i>object</i> ,	Handle of the object
	<i>A_UINT32</i> <i>attribute_id</i> ,	Identifier of the attribute to retrieve
	<i>A_UINT32*</i> <i>a</i> ,	Pointers on the placeholders filled with the attribute fields a and b. Each can be NULL if the corresponding field is not of interest to the caller.
	<i>A_UINT32*</i> <i>b</i>	
)	
Description	<p>The attribute is identified by the argument <code>attribute_id</code>. The precise meaning of this parameter depends on the container type and size and is defined in section 7.9.3.4. Bit [29] of the attribute identifier MUST be set to 1, i.e. it MUST denote a value attribute.</p> <p>They are two kinds of object attributes, which are identified by a bit in their handle value (see Table 7-19):</p> <ul style="list-style-type: none"> Public object attributes can always be extracted whatever the status of the container. Protected attributes can be extracted only if the object's key usage contains the <code>QCOM_CRYPT_USAGE_EXTRACTABLE</code> flag. <p>See section 7.9.3.4 for a definition of all available object attributes and their level of protection.</p>	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.1.6 qcom_crypto_transient_obj_alloc

Definition	Allocate an uninitialized transient object, that is, a container for attributes.	
Prototype	A_CRYPTO_STATUS	
	qcom_crypto_transient_obj_alloc(
	<i>A_UINT32</i> <i>obj_type</i> ,	Type of uninitialized object container to be created (see Table 7-17).

	<i>A_UINT32 max_key_size,</i>	Key Size of the object. Valid values depend on the object type and are defined in Table 7-5 .
	<i>qcom_crypto_obj_hdl_t* object</i>	Filled with a handle on the newly created key container
)	
Description	<p>Transient objects are used to hold a cryptographic object (key or key-pair). The object type and the maximum key size MUST be specified so that all the container resources can be pre-allocated. As allocated, the container is uninitialized. It can be initialized by subsequently importing the object material, generating an object, deriving an object, or loading an object from the Trusted Storage.</p> <p>The initial value of the key usage associated with the container is 0xFFFFFFFF, which means that it contains all usage flags. You can use the function <code>qcom_crypto_obj_usage_restrict</code> to restrict the usage of the container.</p> <p>The returned handle is used to refer to the newly-created container in all subsequent functions that require an object container: key management and operation functions. The handle remains valid until the container is deallocated using the function <code>qcom_crypto_transient_obj_free</code>.</p>	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

As shown in [Table 7-5](#), the object type determines the possible object size to be passed to `qcom_crypto_transient_obj_alloc`, which is not necessarily the size of the object to allocate. In particular, for key objects the size to be passed is the one of the appropriate key sizes described in [Table 7-5](#).

Table 7-5 Object types and key sizes

Object type	Possible key sizes
QCOM_CRYPT0_OBJ_TYPE_AES	128 or 256 bits
QCOM_CRYPT0_OBJ_TYPE_HMAC_MD5	Between 64 and 512 bits, multiple of 8 bits
QCOM_CRYPT0_OBJ_TYPE_HMAC_SHA1	Between 80 and 512 bits, multiple of 8 bits
QCOM_CRYPT0_OBJ_TYPE_HMAC_SHA256	Between 192 and 1024 bits, multiple of 8 bits
QCOM_CRYPT0_OBJ_TYPE_HMAC_SHA384	Between 256 and 1024 bits, multiple of 8 bits
QCOM_CRYPT0_OBJ_TYPE_HMAC_SHA512	Between 256 and 1024 bits, multiple of 8 bits
QCOM_CRYPT0_OBJ_TYPE_RSA_PUBLIC_KEY	The number of bits in the modulus. 256, 512, 768, 1024 are supported
QCOM_CRYPT0_OBJ_TYPE_RSA_KEYPAIR	Same as for RSA public key size.
QCOM_CRYPT0_OBJ_TYPE_DH_KEYPAIR	From 256 to 2048 bits
QCOM_CRYPT0_OBJ_TYPE_ECDSA_PUBLIC_KEY	All the curve sizes defined in Table 3-14 are supported.
QCOM_CRYPT0_OBJ_TYPE_ECDSA_KEYPAIR	MUST be same value as for ECDSA public key size.
QCOM_CRYPT0_OBJ_TYPE_ECDH_KEYPAIR	All the curve sizes defined in Table 3-14 are supported.
QCOM_CRYPT0_OBJ_TYPE_GENERIC_SECRET	Multiple of 8 bits, up to 4096 bits. This type is intended for secret data that is not directly used as a key in a cryptographic operation, but participates in a key derivation.

Object type	Possible key sizes
QCOM_CRYPTOBJ_TYPE_ED25519_PUBLIC_KEY	256 bits
QCOM_CRYPTOBJ_TYPE_ED25519_KEYPAIR	256 bits
QCOM_CRYPTOBJ_TYPE_CURVE25519_KEYPAIR	256 bits
QCOM_CRYPTOBJ_TYPE_SRP_KEYPAIR	1024, 1536, 2048, 3072 bit modulus as defined in RFC 5054

NOTE: Using RSA keys smaller than 1024 bits is nowadays considered insecure but may be required for legacy protocols.

7.9.1.7 qcom_crypto_transient_obj_free

Definition	Deallocate a transient object previously allocated with qcom_crypto_transient_obj_alloc.	
Prototype	A_CRYPTO_STATUS qcom_crypto_transient_obj_free(qcom_crypto_obj_hdl_t object	Handle on the object to free
)	
Description	After this function has been called, the object handle is no longer valid and all resources associated with the transient object are reclaimed. If the object is initialized, the object attributes are cleared before the object is deallocated.	
Return value	A_CRYPTO_SUCCESS / A_CRYPTO_ERROR	

7.9.1.8 qcom_crypto_transient_obj_reset

Definition	Reset a transient object to its initial state after allocation.	
Prototype	A_CRYPTO_STATUS qcom_crypto_transient_obj_reset(qcom_crypto_obj_hdl_t object	Handle on a transient object to reset
)	
Description	If the object is currently initialized, the function clears the object of all its material. The object is then uninitialized again. In any case, the function resets the key usage of the container to 0xFFFFFFFF	
Return value	A_CRYPTO_SUCCESS / A_CRYPTO_ERROR	

7.9.1.9 qcom_crypto_transient_obj_populate

Definition	Populate an uninitialized object container with object attributes passed in the attrs parameter.	
Prototype	A_CRYPTO_STATUS qcom_crypto_transient_obj_populate(qcom_crypto_obj_hdl_t object,	Handle on an already created transient and uninitialized object
	qcom_crypto_attribute_t *attrs, uint32_t attr_count	Array of object attributes
)	

Description	<p>When this function is called, the object MUST be uninitialized. If the object is initialized, the caller MUST first clear it using the function <code>qcom_crypto_transient_obj_reset</code>.</p> <p>Note that if the object type is a key-pair, then this function sets both the private and public parts of the key- pair.</p> <p>As shown in Table 7-6, the interpretation of the <code>attrs</code> parameter depends on the object type. The values of all attributes are copied into the object so that the <code>attrs</code> array and all the memory buffers it points to may be freed after this routine returns without affecting the object. Only the attributes specified in Table 7-6 associated with the object's type are valid.</p>
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR

Table 7-6 Supported attributes

Object Type	Attributes
QCOM_CRYPTO_OBJ_TYPE_AES	For all secret key objects, the QCOM_CRYPTO_ATTR_SECRET_VALUE MUST be provided.
QCOM_CRYPTO_OBJ_TYPE_HMAC_MD5	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA1	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA256	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA384	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA512	
QCOM_CRYPTO_OBJ_TYPE_GENERIC_SECRET	
QCOM_CRYPTO_OBJ_TYPE_RSA_PUBLIC_KEY	The following parts MUST be provided: QCOM_CRYPTO_ATTR_RSA_MODULUS QCOM_CRYPTO_ATTR_RSA_PUBLIC_EXPONENT
QCOM_CRYPTO_OBJ_TYPE_RSA_KEYPAIR	The following parts (including CRT params) MUST be provided: QCOM_CRYPTO_ATTR_RSA_MODULUS QCOM_CRYPTO_ATTR_RSA_PUBLIC_EXPONENT QCOM_CRYPTO_ATTR_RSA_PRIVATE_EXPONENT QCOM_CRYPTO_ATTR_RSA_PRIME1 QCOM_CRYPTO_ATTR_RSA_PRIME2 QCOM_CRYPTO_ATTR_RSA_EXPONENT1 QCOM_CRYPTO_ATTR_RSA_EXPONENT2 QCOM_CRYPTO_ATTR_RSA_COEFFICIENT
QCOM_CRYPTO_OBJ_TYPE_ECDSA_PUBLIC_KEY	The following parts MUST be provided: QCOM_CRYPTO_ATTR_ECC_PUBLIC_VALUE_X QCOM_CRYPTO_ATTR_ECC_PUBLIC_VALUE_Y QCOM_CRYPTO_ATTR_ECC_CURVE
QCOM_CRYPTO_OBJ_TYPE_ECDSA_KEYPAIR	The following parts MUST be provided: QCOM_CRYPTO_ATTR_ECC_PRIVATE_VALUE QCOM_CRYPTO_ATTR_ECC_PUBLIC_VALUE_X QCOM_CRYPTO_ATTR_ECC_PUBLIC_VALUE_Y QCOM_CRYPTO_ATTR_ECC_CURVE

Object Type	Attributes
QCOM_CRYPT0_OBJ_TYPE_ECDH_KEYPAIR	The following parts MUST be provided: QCOM_CRYPT0_ATTR_ECC_PRIVATE_VALUE QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_X QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_Y QCOM_CRYPT0_ATTR_ECC_CURVE
QCOM_CRYPT0_OBJ_TYPE_DH_KEYPAIR	The following parts MUST be provided: QCOM_CRYPT0_ATTR_DH_PRIME QCOM_CRYPT0_ATTR_DH_BASE QCOM_CRYPT0_ATTR_DH_PUBLIC_VALUE QCOM_CRYPT0_ATTR_DH_PRIVATE_VALUE Optionally, QCOM_CRYPT0_ATTR_DH_SUBPRIME may be provided, too.
QCOM_CRYPT0_OBJ_TYPE_ED25519_PUBLIC_KEY	The following parts MUST be provided: QCOM_CRYPT0_ATTR_ED25519_PUBLIC_VALUE
QCOM_CRYPT0_OBJ_TYPE_ED25519_KEYPAIR	The following parts MUST be provided: QCOM_CRYPT0_ATTR_ED25519_PUBLIC_VALUE QCOM_CRYPT0_ATTR_ED25519_PRIVATE_VALUE
QCOM_CRYPT0_OBJ_TYPE_CURVE25519_KEYPAIR	QCOM_CRYPT0_ATTR_CURVE25519_PUBLIC_VALUE QCOM_CRYPT0_ATTR_CURVE25519_PRIVATE_VALUE
QCOM_CRYPT0_OBJ_TYPE_SRP_KEYPAIR	The following parts MUST be provided: QCOM_CRYPT0_ATTR_SRP_TYPE QCOM_CRYPT0_ATTR_SRP_PRIME QCOM_CRYPT0_ATTR_SRP_GEN QCOM_CRYPT0_ATTR_SRP_PUBLIC_VALUE QCOM_CRYPT0_ATTR_SRP_PRIVATE_VALUE In case of server, QCOM_CRYPT0_ATTR_SRP_VERIFIER must be provided

7.9.1.10 qcom_crypto_key_gen

Definition	Generate a random key or a key-pair and populates a transient key object with the generated key material.	
Prototype	A_CRYPT0_STATUS qcom_crypto_key_gen(
	qcom_crypto_obj_hdl_t obj,	Handle on an uninitialized transient key to populate with the generated key

	<i>A_UINT32 keySize,</i>	Requested key size. MUST be less than or equal to the maximum key size specified when the object container was created. MUST be a valid value as defined in Table 7-5 .
	<i>qcom_crypto_attribute_t *params,</i> <i>A_UINT32 paramCount</i>	Parameters for the key generation. The values of all parameters are copied into the object so that the params array and all the memory buffers it points to may be freed after this routine returns without affecting the object.
)	
Description	The size of the desired key is passed in the <i>key_size</i> parameter and MUST be less than or equal to the maximum key size specified when the transient object was created. The valid values for key size are defined in Table 7-5 . As shown in Table 7-7 , the generation algorithm can take parameters depending on the object type.	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

Table 7-7 Object types and key sizes

Object type	Details
QCOM_CRYPTO_OBJ_TYPE_AES	No parameter is necessary. The function generates the attribute QCOM_CRYPTO_ATTR_SECRET_VALUE. The generated value SHALL be the full key size.
QCOM_CRYPTO_OBJ_TYPE_HMAC_MD5	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA1	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA256	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA384	
QCOM_CRYPTO_OBJ_TYPE_HMAC_SHA512	
QCOM_CRYPTO_OBJ_TYPE_GENERIC_SECRET	The following domain parameters MUST be passed to the function: QCOM_CRYPTO_ATTR_DH_PRIME QCOM_CRYPTO_ATTR_DH_BASE The following parameters can optionally be passed: QCOM_CRYPTO_ATTR_DH_SUBPRIME (q): If present, constrains the private value x to be in the range [2, q-2] QCOM_CRYPTO_ATTR_DH_X_BITS (ℓ): If present, constrains the private value x to have ℓ bits If neither of these optional parts is specified, then the only constraint on x is that it is less than p-1. The function generates and populates the following attributes: QCOM_CRYPTO_ATTR_DH_PUBLIC_VALUE QCOM_CRYPTO_ATTR_DH_PRIVATE_VALUE
QCOM_CRYPTO_OBJ_TYPE_DH_KEYPAIR	

Object type	Details
QCOM_CRYPTOBJ_TYPE_ECDSA_KEYPAIR	<p>The following domain parameters MUST be passed to the function:</p> <p>QCOM_CRYPT_ATTR_ECC_CURVE</p> <p>The function generates and populates the following attributes:</p> <p>QCOM_CRYPT_ATTR_ECC_PUBLIC_VALUE_X QCOM_CRYPT_ATTR_ECC_PUBLIC_VALUE_Y QCOM_CRYPT_ATTR_ECC_PRIVATE_VALUE</p>
QCOM_CRYPTOBJ_TYPE_ECDH_KEYPAIR	<p>The following domain parameters MUST be passed to the function:</p> <p>QCOM_CRYPT_ATTR_ECC_CURVE</p> <p>The function generates and populates the following attributes:</p> <p>QCOM_CRYPT_ATTR_ECC_PUBLIC_VALUE_X QCOM_CRYPT_ATTR_ECC_PUBLIC_VALUE_Y QCOM_CRYPT_ATTR_ECC_PRIVATE_VALUE</p>
QCOM_CRYPTOBJ_TYPE_ED25519_KEYPAIR	<p>No domain parameters needed.</p> <p>The function generates and populates the following attributes:</p> <p>QCOM_CRYPT_ATTR_ED25519_PUBLIC_VALUE QCOM_CRYPT_ATTR_ED25519_PRIVATE_VALUE</p>
QCOM_CRYPTOBJ_TYPE_CURVE25519_KEYPAIR	<p>No domain parameters needed.</p> <p>The function generates and populates the following attributes:</p> <p>QCOM_CRYPT_ATTR_CURVE25519_PUBLIC_VALUE QCOM_CRYPT_ATTR_CURVE25519_PRIVATE_VALUE</p>
QCOM_CRYPTOBJ_TYPE_SRP_KEYPAIR	<p>The following domain parameters MUST be passed to the function:</p> <p>QCOM_CRYPT_ATTR_SRP_PRIME QCOM_CRYPT_ATTR_SRP_GEN QCOM_CRYPT_ATTR_SRP_TYPE</p> <p>In case of server,</p> <p>QCOM_CRYPT_ATTR_SRP_VERIFIER QCOM_CRYPT_ATTR_SRP_HASH (optional)</p> <p>If attribute QCOM_CRYPT_ATTR_SRP_HASH is not provided, SHA1 is used as SRP hash function</p> <p>The function generates and populates the following attributes:</p> <p>QCOM_CRYPT_ATTR_SRP_PUBLIC_VALUE QCOM_CRYPT_ATTR_SRP_PRIVATE_VALUE</p>

7.9.2 Cryptographic operations API

Table 7-8 Supported cryptographic algorithms

Algorithm type	Supported algorithm
Digests	MD5 SHA-1 SHA-256 SHA-384 SHA-512
Symmetric ciphers	AES
Message Authentication Codes (MACs)	AES-CMAC HMAC with one of the supported digests
Authenticated Encryption (AE)	AES-CCM with support for Additional Authenticated Data (AAD) AES-GCM with support for Additional Authenticated Data (AAD) AEAD ChaCha20 Poly1305
Asymmetric Encryption Schemes	RSA PKCS1-V1.5
Asymmetric Signature Schemes	RSA PKCS1-V1.5 ECDSA Ed25519
Key Exchange Algorithms	Diffie-Hellman ECDH Curve25519 SRP

Table 7-9 ECC cryptographic algorithms

Algorithm type	Supported algorithm
Asymmetric Signature Schemes	ECDSA Ed25519
Key Exchange Algorithms	ECDH Curve25519

7.9.2.1 Data types

QCOM_CRYPTO_MODE

The enumeration QCOM_CRYPTO_MODE lists the modes for all the cryptographic operations.

```
typedef enum {
    QCOM_CRYPTO_MODE_ENCRPT = 0,
    QCOM_CRYPTO_MODE_DECRYPT = 1,
    QCOM_CRYPTO_MODE_SIGN = 2,
    QCOM_CRYPTO_MODE_VERIFY = 3,
    QCOM_CRYPTO_MODE_MAC = 4,
    QCOM_CRYPTO_MODE_DIGEST = 5,
    QCOM_CRYPTO_MODE_DERIVE = 6
} QCOM_CRYPTO_MODE;
```

Table 7-10 Possible QCOM_CRYPT0_MODE values

Name	Comment
QCOM_CRYPT0_MODE_ENCRYPT	Encryption mode
QCOM_CRYPT0_MODE_DECRYPT	Decryption mode
QCOM_CRYPT0_MODE_SIGN	Signature generation mode
QCOM_CRYPT0_MODE_VERIFY	Signature verification mode
QCOM_CRYPT0_MODE_MAC	MAC mode
QCOM_CRYPT0_MODE_DIGEST	Digest mode
QCOM_CRYPT0_MODE_DERIVE	Key derivation mode

qcom_op_info_t

```
typedef struct {
    A_UINT32 algorithm;
    A_UINT32 mode;
    A_UINT32 digest_len;
    A_UINT32 max_key_size;
    A_UINT32 key_size;
    A_UINT32 handle_state;
} qcom_op_info_t;
```

qcom_crypto_op_hdl_t

qcom_crypto_op_hdl_t is an opaque handle on a cryptographic operation.

```
typedef A_UINT32 qcom_crypto_op_hdl_t;
```

7.9.2.2 qcom_crypto_op_alloc

Definition	Allocates a handle for a new cryptographic operation and sets the mode and algorithm type. If this function does not return with A_CRYPT0_SUCESS then there is no valid handle value.	
Prototype	A_CRYPT0_STATUS qcom_crypto_op_alloc(qcom_crypto_op_hdl_t *operation, A_UINT32 algorithm, A_UINT32 mode, A_UINT32 maxKeySize)	Reference to generated operation handle
		One of the algorithms enumerated in section 7.9.3.1
		The operation mode MUST be one of the constants defined in section 7.9.2.1 . It MUST be compatible with the algorithm as defined by Table 7-11 .
		Maximum key size in bits for the operation – must be a valid value for the algorithm as defined in Table 7-5 .
Return value	A_CRYPT0_SUCESS / A_CRYPT0_ERROR	

Table 7-11 qcom_crypto_op_alloc allowed modes

Algorithm	Possible modes
QCOM_CRYPT0_ALG_AES_CBC_NOPAD	QCOM_CRYPT0_MODE_ENCRYPT QCOM_CRYPT0_MODE_DECRYPT
QCOM_CRYPT0_ALG_AES_CTR	
QCOM_CRYPT0_ALG_AES_CCM	
QCOM_CRYPT0_ALG_AES_GCM	
QCOM_CRYPT0_ALG_CHACHA20_POLY1305	
QCOM_CRYPT0_ALG_AES_CMAC	QCOM_CRYPT0_MODE_MAC
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA1	QCOM_CRYPT0_MODE_SIGN QCOM_CRYPT0_MODE_VERIFY
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA256	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA384	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA512	
QCOM_CRYPT0_ALG_ED25519	
QCOM_CRYPT0_ALG_ECDSA	QCOM_CRYPT0_MODE_ENCRYPT QCOM_CRYPT0_MODE_DECRYPT
QCOM_CRYPT0_ALG_RSAES_PKCS1_V1_5	
QCOM_CRYPT0_ALG_RSA_NOPAD	QCOM_CRYPT0_MODE_DERIVE
QCOM_CRYPT0_ALG_DH_DERIVE_SHARED_SECRET	
QCOM_CRYPT0_ALG_SRP_DERIVE_SHARED_SECRET	
QCOM_CRYPT0_ALG_CURVE25519_DERIVE_SHARED_SECRET	
QCOM_CRYPT0_ALG_ECDH_DERIVE_SHARED_SECRET	QCOM_CRYPT0_MODE_DIGEST
QCOM_CRYPT0_ALG_MD5	
QCOM_CRYPT0_ALG_SHA1	
QCOM_CRYPT0_ALG_SHA256	
QCOM_CRYPT0_ALG_SHA384	
QCOM_CRYPT0_ALG_SHA512	QCOM_CRYPT0_MODE_MAC
QCOM_CRYPT0_ALG_HMAC_MD5	
QCOM_CRYPT0_ALG_HMAC_SHA1	
QCOM_CRYPT0_ALG_HMAC_SHA256	
QCOM_CRYPT0_ALG_HMAC_SHA384	
QCOM_CRYPT0_ALG_HMAC_SHA512	

7.9.2.3 qcom_crypto_op_free

Definition	Deallocate all resources associated with an operation handle. After this function is called, the operation handle is no longer valid. All cryptographic material in the operation is destroyed.	
Prototype	A_CRYPT0_STATUS qcom_crypto_op_free(
	<i>qcom_crypto_op_hdl_t op</i>	Reference to operation handle
)	
Return value	A_CRYPT0_SUCESS / A_CRYPT0_ERROR	

7.9.2.4 qcom_crypto_op_info_get

Definition	Return information about an operation handle.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_info_get(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle on the operation
	<i>qcom_crypto_op_info_t *op_info</i>	Pointer to a structure filled with the operation information
)	
Description	<p>It fills the following fields in the structure operationInfo (defined in section 7.9.2.1):</p> <ul style="list-style-type: none"> ▪ algorithm, mode, max_key_size: The parameters passed to the function qcom_crypto_op_alloc ▪ key_size: If a key is programmed in the operation, the actual size of this key. ▪ digestLength: For a MAC, AE, or Digest, describes the number of bytes in the digest or tag. ▪ handleState: A bit vector describing the current state of the operation. Can contain any combination of the following flags or 0 if no flags are appropriate: <ul style="list-style-type: none"> ▫ QCOM_CRYPTO_HANDLE_FLAG_KEY_SET: Set if the operation key has been set. Always set for digest operations. ▫ QCOM_CRYPTO_HANDLE_FLAG_INITIALIZED: For multi-stage operations, that is, all but QCOM_CRYPTO_OPERATION_ASYMMETRIC_XXX operation classes, whether the operation has been initialized using one of the qcom_crypto_op_XXX_init functions. This flag is always set for Digest operations. 	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.5 qcom_crypto_op_reset

Definition	Reset the operation to initial state before initialization, but after the key has been set.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_reset(
	<i>qcom_crypto_op_hdl_t operation</i>	Handle on the operation
)	
Description	<p>This function can be called on any operation and at any time after the key is set, but is meaningful only for the multi-stage operations, that is, symmetric ciphers, MACs, AEs, and digests.</p> <p>When such a multi-stage operation is active, that is, when it has been initialized but not yet successfully finalized, then the operation is reset to initial state. The operation key(s) are not cleared.</p>	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.6 qcom_crypto_op_key_set

Definition	Program the key of an operation, that is, it associates an operation with a key.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_key_set(
	<i>qcom_crypto_op_hdl_t operation,</i>	Operation handle
	<i>qcom_crypto_obj_hdl_t key</i>	A handle on a key object
)	

Description	<p>The key material is copied from the key object handle into the operation. After the key has been set, there is no longer any link between the operation and the key object. The object handle can be closed or reset and this will not affect the operation. This copied material exists until the operation is freed using <code>qcom_crypto_op_free</code> or another key is set into the operation.</p> <p>The operation MUST be in initial state before the operation and remains in initial state afterwards.</p> <p>The key object type and size MUST be compatible with the type and size of the operation. The operation mode MUST be compatible with key usage:</p> <ul style="list-style-type: none"> ▪ In general, the operation mode MUST be allowed in the object usage. ▪ For the <code>QCOM_CRYPT0_ALG_ALG_RSA_NOPAD</code> algorithm: <ul style="list-style-type: none"> ▫ The only supported modes are <code>QCOM_CRYPT0_MODE_ENCRYPT</code> and <code>QCOM_CRYPT0_MODE_DECRYPT</code>. ▫ For <code>QCOM_CRYPT0_MODE_ENCRYPT</code>, the object usage MUST contain both the <code>QCOM_CRYPT0_USAGE_ENCRYPT</code> and <code>QCOM_CRYPT0_USAGE_VERIFY</code> flags. ▫ For <code>QCOM_CRYPT0_MODE_DECRYPT</code>, the object usage MUST contain both the <code>QCOM_CRYPT0_USAGE_DECRYPT</code> and <code>QCOM_CRYPT0_USAGE_SIGN</code> flags.
Return value	<code>A_CRYPT0_SUCESS</code> / <code>A_CRYPT0_ERROR</code>

For a public key object, the allowed operation modes depend on the type of key and are specified in [Table 7-12](#).

Table 7-12 Public key allowed modes

Key type	Allowed operation modes
<code>QCOM_CRYPT0_OBJ_TYPE_RSA_PUBLIC_KEY</code>	<code>QCOM_CRYPT0_MODE_VERIFY</code> or <code>QCOM_CRYPT0_MODE_ENCRYPT</code>
<code>QCOM_CRYPT0_OBJ_TYPE_ED25519_PUBLIC_KEY</code>	<code>QCOM_CRYPT0_MODE_VERIFY</code>
<code>QCOM_CRYPT0_OBJ_TYPE_ECDSA_PUBLIC_KEY</code>	<code>QCOM_CRYPT0_MODE_VERIFY</code>

If the object is a key-pair then the key parts used in the operation depend on the operation mode as defined in [Table 7-13](#).

Table 7-13 Key-Pair parts for operation modes

Operation mode	Key parts used
<code>QCOM_CRYPT0_MODE_VERIFY</code>	Public
<code>QCOM_CRYPT0_MODE_SIGN</code>	Private
<code>QCOM_CRYPT0_MODE_ENCRYPT</code>	Public
<code>QCOM_CRYPT0_MODE_DECRYPT</code>	Private
<code>QCOM_CRYPT0_MODE_DERIVE</code>	Public and Private

7.9.2.7 `qcom_crypto_op_copy`

Definition	Copy an operation state from one operation handle into another operation handle. This also copies the key material associated with the source operation.	
Prototype	<code>A_CRYPT0_STATUS qcom_crypto_op_copy(</code>	
	<code>qcom_crypto_op_hdl_t dstOperation,</code>	Handle on the destination operation
	<code>qcom_crypto_op_hdl_t srcOperation</code>	Handle on the source operation

)	
Description	<p>The state of srcOperation including the key material currently set up is copied into dstOperation. This function is useful in the following use cases:</p> <ul style="list-style-type: none"> ▪ “Forking” a digest operation after feeding some amount of initial data ▪ Computing intermediate digests <p>The algorithm and mode of dstOperation MUST be equal to the algorithm and mode of srcOperation. The state of srcOperation (initial/active) is copied to dstOperation. If srcOperation has no key programmed, then the key in dstOperation is cleared. If there is a key programmed in srcOperation, then the maximum key size of dstOperation MUST be greater than or equal to the actual key size of srcOperation.</p>	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.8 qcom_crypto_op_digest_update

Definition	Accumulate message data for hashing.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_digest_update(
	qcom_crypto_op_hdl_t operation,	Handle of a running Message Digest operation
	void* chunk, uint32_t chunkSize	Chunk of data to be hashed
)	
Description	The message does not have to be block aligned. Subsequent calls to this function are possible.	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.9 qcom_crypto_op_digest_final

Definition	Finalize the message digest operation and produces the message hash. Afterwards the Message Digest operation is reset to initial state and can be reused.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_digest_final(
	qcom_crypto_op_hdl_t operation,	Handle of a running Message Digest operation
	void* chunk, uint32_t chunkLen,	Last chunk of data to be hashed
	void* hash, uint32_t *hashLen	Output buffer filled with the message hash
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.10 qcom_crypto_op_cipher_init

Definition	Start the symmetric cipher operation. The operation MUST have been associated with a key.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_cipher_init(
	qcom_crypto_op_hdl_t operation,	A handle on an opened cipher operation setup with a key

	<i>void*</i> <i>IV, uint32_t IVLen</i>	Buffer containing the operation Initialization Vector or the initial counter value as appropriate
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.11 qcom_crypto_op_cipher_update

Definition	Encrypt or decrypt input data.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_cipher_update(<i>qcom_crypto_op_hdl_t operation,</i>	
	<i>void*</i> <i>srcData,</i> <i>uint32_t srcLen,</i>	Handle of a running Cipher operation
	<i>void*</i> <i>destData, uint32_t *destLen</i>	Input data buffer to be encrypted or decrypted
)	Output buffer
Description	Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to qcom_crypto_op_cipher_dofinal.	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	
Comments	For the "*_NOPAD" symmetric algorithms, it is the responsibility of the Application to do the padding, so input data must be a multiple of block size. For the QCOM_CRYPTO_ALG_AES_CTR algorithm, if one of the chunks supplied to the qcom_crypto_op_cipher_update function was not a multiple of 16 bytes, the destination buffer length have to be at least (ceil(srcLen/16) * 16) bytes long.	

7.9.2.12 qcom_crypto_op_cipher_dofinal

Definition	Finalize the cipher operation, processing data that has not been processed by previous calls to qcom_crypto_cipher_update as well as data supplied in srcData. The operation handle can be reused or re-initialized.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_cipher_dofinal (<i>qcom_crypto_op_hdl_t operation,</i>	
	<i>void*</i> <i>srcData,</i> <i>uint32_t srcLen,</i>	Handle of a running Cipher operation
	<i>void*</i> <i>destData, uint32_t *destLen</i>	Reference to final chunk of input data to be encrypted or decrypted
)	Output buffer. It can be omitted if the output is to be discarded, e.g. because it is known to be empty.
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	
Comments	For the "*_NOPAD" symmetric algorithms, it is the responsibility of the Application to do the padding, so input data must be a multiple of block size. For the QCOM_CRYPTO_ALG_AES_CTR algorithm, if one of the chunks supplied to the qcom_crypto_op_cipher_update function was not a multiple of 16 bytes, the destination buffer length have to be at least (ceil(srcLen/16) * 16) bytes long.	

7.9.2.13 qcom_crypto_op_mac_init

Definition	Initialize a MAC operation.	
Prototype	void qcom_crypto_op_mac_init(
	<i>qcom_crypto_op_hdl_t operation,</i>	Operation handle
	<i>void* IV, uint32_t IVLen</i>	Input buffer containing the operation Initialization Vector, if applicable
)	
Description	The operation MUST have been associated with a key. If the MAC algorithm does not require an IV, the parameters IV, IVLen are ignored.	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.14 qcom_crypto_op_mac_update

Definition	Accumulate data for a MAC calculation.	
Prototype	void qcom_crypto_op_mac_update(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle of a running MAC operation
	<i>void* chunk, uint32_t chunkSize</i>	Chunk of the message to be MACed
)	
Description	Input data does not have to be a multiple of the block size. Subsequent calls to this function are possible.	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.15 qcom_crypto_op_mac_compute_final

Definition	Finalize the MAC operation with a last chunk of message, and computes the MAC. Afterwards the operation handle can be reused or re-initialized with a new key.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_mac_compute_final(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle of a MAC operation
	<i>void* message, uint32_t messageLen,</i>	Input buffer containing a last message chunk to MAC
	<i>void* mac, uint32_t *macLen</i>	Output buffer filled with the computed MAC
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.16 qcom_crypto_op_mac_compare_final

Definition	Finalize the MAC operation and compares the MAC with the buffer passed to the function. Afterwards the operation handle can be reused and initialized with a new key.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_mac_compare_final(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle of a MAC operation
	<i>void* message, uint32_t messageLen,</i>	Input buffer containing the last message chunk to MAC

	<i>void* mac, uint32_t macLen</i>	Input buffer containing the MAC to check
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR_MAC_INVALID	

7.9.2.17 qcom_crypto_op_ae_init

Definition	Initialize an Authentication Encryption operation. The operation must be initial state and remains in the initial state afterwards.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_ae_init(
	<i>qcom_crypto_op_hdl_t operation,</i>	A handle on the operation
	<i>void* nonce, uint32_t nonceLen,</i>	The operation nonce or IV For AES-GCM, must be 12
	<i>uint32_t tagLen,</i>	Size in bits of the tag <ul style="list-style-type: none"> For AES-GCM, must be 128 For AES-CCM, can be 128, 112, 96, 80, 64, 48, or 32
	<i>uint32_t AADLen,</i>	Length in bytes of the AAD Used only for AES-CCM. Ignored for AES-GCM.
	<i>uint32_t payloadLen</i>	Length in bytes of the payload Used only for AES-CCM. Ignored for AES-GCM.
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.18 qcom_crypto_op_ae_aad_update

Definition	Feed a new chunk of Additional Authentication Data (AAD) to the AE operation.	
Prototype	void qcom_crypto_op_ae_aad_update(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle on the AE operation
	<i>void* AADdata, uint32_t AADdataLen</i>	Input buffer containing the chunk of AAD
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	
Comments	Subsequent calls to this function are available ONLY for the QCOM_CRYPTO_ALG_CHACHA20_POLY1305 algorithm.	

7.9.2.19 qcom_crypto_op_ae_update

Definition	Accumulate data for an Authentication Encryption operation.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_ae_update(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle of a running AE operation
	<i>void* srcData, uint32_t srcLen,</i>	Input data buffer to be encrypted or decrypted
	<i>void* destData, uint32_t *destLen</i>	Output buffer
)	

Description	Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated.
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR_SHORT_BUFFER
Comments	This function is ONLY supported for QCOM_CRYPT0_ALG_CHACHA20_POLY1305 algorithm.

7.9.2.20 qcom_crypto_op_ae_encrypt_final

Definition	Process data that has not been processed by previous calls to qcom_crypto_ae_update as well as data supplied in srcData. It completes the AE operation and computes the tag. The operation handle can be reused or newly initialized.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_ae_encrypt_final(qcom_crypto_op_hdl_t operation, void* srcData, uint32_t srcLen, void* destData, uint32_t *destLen, void* tag, uint32_t *tagLen)	
	qcom_crypto_op_hdl_t operation,	Handle of a running AE operation
	void* srcData, uint32_t srcLen,	Reference to final chunk of input data to be encrypted
	void* destData, uint32_t *destLen,	Output buffer. Can be omitted if the output is to be discarded, e.g. because it is known to be empty.
	void* tag, uint32_t *tagLen	Output buffer filled with the computed tag
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

7.9.2.21 qcom_crypto_op_ae_decrypt_final

Definition	Process data that has not been processed by previous calls to qcom_crypto_ae_update as well as data supplied in srcData. It completes the AE operation and compares the computed tag with the tag supplied in the parameter tag.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_ae_decrypt_final(qcom_crypto_op_hdl_t operation, void* srcData, uint32_t srcLen, void* destData, uint32_t *destLen, void* tag, uint32_t tagLen)	
	qcom_crypto_op_hdl_t operation,	Handle of a running AE operation
	void* srcData, uint32_t srcLen,	Reference to final chunk of input data to be decrypted
	void* destData, uint32_t *destLen,	Output buffer. Can be omitted if the output is to be discarded, e.g. because it is known to be empty.
	void* tag, uint32_t tagLen	Input buffer containing the tag to compare
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR / A_CRYPTO_ERROR_MAC_INVALID	

7.9.2.22 qcom_crypto_op_asymmetric_encrypt

Definition	Encrypt a message within an asymmetric operation.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_asymmetric_encrypt(

	<i>qcom_crypto_op_hdl_t operation,</i>	Handle on the operation, which MUST have been suitably set up with an operation key
	<i>qcom_crypto_attribute_t* params, uint32_t paramCount</i>	Optional operation parameters
	<i>void* srcData, uint32_t srcLen,</i>	Input buffer
	<i>void* destData, uint32_t *destLen</i>	Output buffer
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	
Comments	This function can be called only with an operation of the following algorithms: <ul style="list-style-type: none"> ▪ QCOM_CRYPTO_ALG_RSAES_PKCS1_V1_5 ▪ QCOM_CRYPTO_ALG_RSA_NOPAD 	

7.9.2.23 qcom_crypto_op_asymmetric_decrypt

Definition	Decrypt a message within an asymmetric operation.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_asymmetric_decrypt(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle on the operation, which MUST have been suitably set up with an operation key
	<i>qcom_crypto_attribute_t* params, uint32_t paramCount</i>	Optional operation parameters
	<i>void* srcData, uint32_t srcLen,</i>	Input buffer
	<i>void* destData, uint32_t *destLen</i>	Output buffer
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	
Comments	This function can be called only with an operation of the following algorithms: <ul style="list-style-type: none"> ▪ QCOM_CRYPTO_ALG_RSAES_PKCS1_V1_5 ▪ QCOM_CRYPTO_ALG_RSA_NOPAD 	

7.9.2.24 qcom_crypto_op_sign_digest

Definition	Sign a message digest within an asymmetric operation.	
Prototype	A_CRYPTO_STATUS qcom_crypto_op_sign_digest(
	<i>qcom_crypto_op_hdl_t operation,</i>	Handle on the operation, which MUST have been suitably set up with an operation key
	<i>qcom_crypto_attr_t* params, uint32_t paramCount,</i>	Optional operation parameters
	<i>void* digest, uint32_t digestLen,</i>	Input buffer containing the input message digest
	<i>void* signature, uint32_t *signatureLen</i>	Output buffer written with the signature of the digest
)	
Return value	A_CRYPTO_SUCESS / A_CRYPTO_ERROR	

Comments	<p>Only an already hashed message can be signed.</p> <p>This function can be called only with an operation of the following algorithms:</p> <ul style="list-style-type: none"> ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA1 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA256 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA384 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA512 ▪ QCOM_CRYPT0_ALG_ECDSA_P192 ▪ QCOM_CRYPT0_ALG_ECDSA_P224 ▪ QCOM_CRYPT0_ALG_ECDSA_P256 ▪ QCOM_CRYPT0_ALG_ECDSA_P384 ▪ QCOM_CRYPT0_ALG_ECDSA_P521 ▪ QCOM_CRYPT0_ALG_ED25519 <p>The parameters params, paramCount contain the operation parameters which may be needed for algorithms added in the future. Currently, these parameters are ignored.</p> <p>Where a hash algorithm is specified in the algorithm, digestLen SHALL be equal to the digest length of this hash algorithm.</p>
-----------------	---

7.9.2.25 qcom_crypto_op_verify_digest

Definition	Verify a message digest signature within an asymmetric operation.	
Prototype	A_CRYPT0_STATUS qcom_crypto_op_verify_digest(<i>qcom_crypto_op_hdl_t</i> operation, <i>qcom_crypto_attr_t*</i> params, <i>uint32_t</i> paramCount, <i>void*</i> digest, <i>uint32_t</i> digestLen, <i>void*</i> signature, <i>uint32_t</i> signatureLen)	
		Handle on the operation, which MUST have been suitably set up with an operation key
		Optional operation parameters
		Input buffer containing the input message digest
		Input buffer containing the signature to verify
Return value	A_CRYPT0_SUCESS / A_CRYPT0_ERROR	
Comments	<p>Only an already hashed message can be signed.</p> <p>This function can be called only with an operation of the following algorithms:</p> <ul style="list-style-type: none"> ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA1 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA256 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA384 ▪ QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA512 ▪ QCOM_CRYPT0_ALG_ECDSA_P192 ▪ QCOM_CRYPT0_ALG_ECDSA_P224 ▪ QCOM_CRYPT0_ALG_ECDSA_P256 ▪ QCOM_CRYPT0_ALG_ECDSA_P384 ▪ QCOM_CRYPT0_ALG_ECDSA_P521 ▪ QCOM_CRYPT0_ALG_ED25519 <p>The parameters params, paramCount contain the operation parameters which may be needed for algorithms added in the future. Currently, these parameters are ignored.</p> <p>Where a hash algorithm is specified in the algorithm, digestLen SHALL be equal to the digest length of this hash algorithm.</p>	

7.9.2.26 qcom_crypto_key_derive

Definition	<p>The <code>qcom_crypto_key_derive</code> function can only be used with algorithms defined in Table 7-14.</p> <p>The parameters <code>params</code>, <code>paramCount</code> contain the operation parameters listed in Table 7-14.</p> <p>The <code>derivedKey</code> handle MUST refer to an object with type <code>QCOM_CRYPT0_OBJ_TYPE_GENERIC_SECRET</code>.</p> <p>On completion the derived key is placed into the <code>QCOM_CRYPT0_ATTR_SECRET_VALUE</code> attribute of the <code>derivedKey</code> handle.</p>	
Prototype	<code>A_CRYPT0_STATUS</code> <code>qcom_crypto_key_derive(</code>	
	<code>qcom_crypto_op_hdl_t operation,</code>	Handle on the operation, which MUST have been suitably set up with an operation key
	<code>qcom_crypto_attribute_t* params,</code> <code>uint32_t paramCount</code>	Operation parameters
	<code>qcom_crypto_obj_hdl_t derivedKey</code>	Handle on an uninitialized transient object filled with the derived key
	<code>)</code>	
Return value	<code>A_CRYPT0_SUCESS</code> / <code>A_CRYPT0_ERROR</code>	

Table 7-14 Asymmetric derivation operation parameters

Algorithm	Possible Operation Parameters
<code>QCOM_CRYPT0_ALG_DH_DERIVE_SHARED_SECRET</code>	<code>QCOM_CRYPT0_ATTR_DH_PUBLIC_VALUE</code> : Public key part of the other party. This parameter is mandatory.
<code>QCOM_CRYPT0_ALG_ECDH_NIST_P192_DERIVE_SHARED_SECRET</code> <code>QCOM_CRYPT0_ALG_ECDH_NIST_P224_DERIVE_SHARED_SECRET</code> <code>QCOM_CRYPT0_ALG_ECDH_NIST_P256_DERIVE_SHARED_SECRET</code> <code>QCOM_CRYPT0_ALG_ECDH_NIST_P384_DERIVE_SHARED_SECRET</code> <code>QCOM_CRYPT0_ALG_ECDH_NIST_P521_DERIVE_SHARED_SECRET</code>	<code>QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_X</code> , <code>QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_Y</code> : Public key part of the other party. This parameter is mandatory.
<code>QCOM_CRYPT0_ALG_CURVE25519_DERIVE_SHARED_SECRET</code>	<code>QCOM_CRYPT0_ATTRIB_CURVE25519_PUBLIC_VALUE</code> . This parameter is mandatory

Algorithm	Possible Operation Parameters
QCOM_CRYPT0_ALG_SRP_DERIVE_SHARED_SEC RET	QCOM_CRYPT0_ATTRIB_SRP_PUBLIC_V ALUE QCOM_CRYPT0_ATTRIB_SRP_TYPE Additional attributes for client: QCOM_CRYPT0_ATTRIB_SRP_PASSWOR D QCOM_CRYPT0_ATTRIB_SRP_USERNAM E QCOM_CRYPT0_ATTRIB_SRP_SALT QCOM_CRYPT0_ATTRIB_SRP_HASH (optional) If attribute QCOM_CRYPT0_ATTRIB_SRP_HASH is not provided, SHA1 is used as SRP hash function

7.9.2.27 qcom_crypto_rng_get

Definition	Retrieve 'len' random bytes into *ptr. Uses hardware RNG.	
Prototype	A_CRYPT0_STATUS qcom_crypto_rng_get (
	void *buffer,	Pointer to buffer to return random bytes
	A_UINT16 len	Number of random bytes to return
)	
Return value	A_CRYPT0_SUCESS / A_CRYPT0_ERROR	

7.9.3 Cryptographic algorithms specification

This section specifies the cryptographic algorithms, key types, and key parts supported in the Cryptographic Operations API.

NOTE: For the “NOPAD” symmetric algorithms, it is the responsibility of the TA to do the padding.

7.9.3.1 List of algorithm identifiers

[Table 7-15](#) provides an exhaustive list of all algorithm identifiers specified in the Cryptographic Operations API.

Table 7-15 List of algorithm identifiers

Name	Identifier	Comments
QCOM_CRYPT0_ALG_AES_CBC_NOPAD	0x10000110	
QCOM_CRYPT0_ALG_AES_CTR	0x10000210	The counter MUST be encoded as a 16-byte buffer in big-endian form. Between two consecutive blocks, the counter MUST be incremented by 1. If it reaches the limit of all 128 bits set to 1, it MUST wrap around to 0.

Name	Identifier	Comments
QCOM_CRYPT0_ALG_AES_CMAC	0x30000610	
QCOM_CRYPT0_ALG_AES_CCM	0x40000710	
QCOM_CRYPT0_ALG_AES_GCM	0x40000810	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA1	0x70002830	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA256	0x70004830	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA384	0x70005830	
QCOM_CRYPT0_ALG_RSASSA_PKCS1_V1_5_SHA512	0x70006830	
QCOM_CRYPT0_ALG_RSAES_PKCS1_V1_5	0x60000130	
QCOM_CRYPT0_ALG_RSA_NOPAD	0x60000030	
QCOM_CRYPT0_ALG_DH_DERIVE_SHARED_SECRET	0x80000032	
QCOM_CRYPT0_ALG_MD5	0x50000001	
QCOM_CRYPT0_ALG_SHA1	0x50000002	
QCOM_CRYPT0_ALG_SHA256	0x50000004	
QCOM_CRYPT0_ALG_SHA384	0x50000005	
QCOM_CRYPT0_ALG_SHA512	0x50000006	
QCOM_CRYPT0_ALG_HMAC_MD5	0x30000001	
QCOM_CRYPT0_ALG_HMAC_SHA1	0x30000002	
QCOM_CRYPT0_ALG_HMAC_SHA256	0x30000004	
QCOM_CRYPT0_ALG_HMAC_SHA384	0x30000005	
QCOM_CRYPT0_ALG_HMAC_SHA512	0x30000006	
QCOM_CRYPT0_ALG_ECDSA_P192	0x70001042	
QCOM_CRYPT0_ALG_ECDSA_P224	0x70002042	
QCOM_CRYPT0_ALG_ECDSA_P256	0x70003042	
QCOM_CRYPT0_ALG_ECDSA_P384	0x70004042	
QCOM_CRYPT0_ALG_ECDSA_P521	0x70005042	
QCOM_CRYPT0_ALG_ECDH_P192	0x80001042	
QCOM_CRYPT0_ALG_ECDH_P224	0x80002042	
QCOM_CRYPT0_ALG_ECDH_P256	0x80003042	
QCOM_CRYPT0_ALG_ECDH_P384	0x80004042	
QCOM_CRYPT0_ALG_ECDH_P521	0x80005042	
QCOM_CRYPT0_ALG_ED25519	0x700000C0	
QCOM_CRYPT0_ALG_CURVE25519_DERIVE_SHA256_SECRET	0x800000C1	
QCOM_CRYPT0_ALG_SRP_DERIVE_SHARED_SECRET	0x800000C1	
QCOM_CRYPT0_ALG_CHACHA20_POLY1305	0x400000C3	

Table 7-16 Structure of algorithm identifier

Bits	Function	Values
Bits [31:28]	Specify the algorithm class and determine which function can be called	0x1: Block cipher 0x3: MAC 0x4: Authenticated Encryption cipher 0x5: Digest 0x6: Asymmetric cipher 0x7: Asymmetric signature 0x8: Key derivation 0xA: Object handle
Bits [7:0]	Identify the underlying main algorithm itself	0x00: Generic Secret Key object 0x01: MD5 0x02: SHA-1 0x04: SHA-256 0x05: SHA-384 0x06: SHA-512 0x10: AES 0x30: RSA 0x32: DH 0x41: ECDSA 0x42: ECDH 0xBF: No appropriate algorithm 0xC0: Ed25519 0xC1: Curve25519 0xC2: SRP 0xC3: ChaCha20
Bits [11:8]	Define the chaining mode or padding	
Bits [15:12]	Define the message digest for asymmetric signature algorithms	
Bits [19:16]	Define the elliptic curve used in ECC methods if supported	
Bits [27:20]	Not used	

7.9.3.2 Object types

Object handles are a special class of algorithm handle and follow the rules in [Table 7-16](#) but only use the object type and algorithm fields.

Table 7-17 List of object types

Name	Identifier
QCOM_CRYPTOBJ_TYPE_AES	0xA0000010
QCOM_CRYPTOBJ_TYPE_HMAC_MD5	0xA0000001
QCOM_CRYPTOBJ_TYPE_HMAC_SHA1	0xA0000002
QCOM_CRYPTOBJ_TYPE_HMAC_SHA256	0xA0000004
QCOM_CRYPTOBJ_TYPE_HMAC_SHA384	0xA0000005
QCOM_CRYPTOBJ_TYPE_HMAC_SHA512	0xA0000006
QCOM_CRYPTOBJ_TYPE_RSA_PUBLIC_KEY	0xA0000030
QCOM_CRYPTOBJ_TYPE_RSA_KEYPAIR	0xA1000030

Name	Identifier
QCOM_CRYPT0_OBJ_TYPE_DH_KEYPAIR	0xA1000032
QCOM_CRYPT0_OBJ_TYPE_ECDSA_PUBLIC_KEY	0xA0000041
QCOM_CRYPT0_OBJ_TYPE_ECDSA_KEYPAIR	0xA1000041
QCOM_CRYPT0_OBJ_TYPE_ECDH_KEYPAIR	0xA1000042
QCOM_CRYPT0_OBJ_TYPE_GENERIC_SECRET	0xA0000000
QCOM_CRYPT0_OBJ_TYPE_ED25519_PUBLIC_KEY	0xA00000C0
QCOM_CRYPT0_OBJ_TYPE_ED25519_KEYPAIR	0xA10000C0
QCOM_CRYPT0_OBJ_TYPE_CURVE25519_KEYPAIR	0xA10000C1
QCOM_CRYPT0_OBJ_TYPE_SRP_KEYPAIR	0xA00000C2
QCOM_CRYPT0_OBJ_TYPE_CHACHA20	0xA00000C3

Object types using implementation-specific algorithms are defined by the implementation.

7.9.3.3 Elliptic curve types

The elliptic curve cryptography (ECC) curves that are supported is given in [Table 7-18](#). All curve definitions come from <http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf> [NIST Re Cur].

Table 7-18 List of supported ECC curves

Name
QCOM_CRYPT0_ECC_CURVE_NIST_P192
QCOM_CRYPT0_ECC_CURVE_NIST_P224
QCOM_CRYPT0_ECC_CURVE_NIST_P256
QCOM_CRYPT0_ECC_CURVE_NIST_P384
QCOM_CRYPT0_ECC_CURVE_NIST_P521

7.9.3.4 Object or operation attributes

Table 7-19 Object or operation attributes

Name	Value	Protection	Type	Format (Table 7-20)	Comment
QCOM_CRYPT0_ATTR_SECRET_VALUE	0xC0000000	Protected	Ref	binary	Used for all secret keys for symmetric ciphers, MACs, and HMACs
QCOM_CRYPT0_ATTR_RSA_MODULUS	0xD0000130	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_RSA_PUBLIC_EXPONENT	0xD0000230	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_RSA_PRIVATE_EXPONENT	0xC0000330	Protected	Ref	bignum	

Name	Value	Protection	Type	Format (Table 7-20)	Comment
QCOM_CRYPT0_ATTR_RSA_PRIME1	0xC0000430	Protected	Ref	bignum	Usually referred to as p.
QCOM_CRYPT0_ATTR_RSA_PRIME2	0xC0000530	Protected	Ref	bignum	q
QCOM_CRYPT0_ATTR_RSA_EXPONENT1	0xC0000630	Protected	Ref	bignum	dp
QCOM_CRYPT0_ATTR_RSA_EXPONENT2	0xC0000730	Protected	Ref	bignum	dq
QCOM_CRYPT0_ATTR_RSA_COEFFICIENT	0xC0000830	Protected	Ref	bignum	iq
QCOM_CRYPT0_ATTR_DH_PRIME	0xD0001032	Public	Ref	bignum	p
QCOM_CRYPT0_ATTR_DH_SUBPRIME	0xD0001132	Public	Ref	bignum	q
QCOM_CRYPT0_ATTR_DH_BASE	0xD0001232	Public	Ref	bignum	g
QCOM_CRYPT0_ATTR_DH_X_BITS	0xF0001332	Public	Value	int	<i>ℓ</i>
QCOM_CRYPT0_ATTR_DH_PUBLIC_VALUE	0xD0000132	Public	Ref	bignum	y
QCOM_CRYPT0_ATTR_DH_PRIVATE_VALUE	0xC0000232	Protected	Ref	bignum	x
QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_X	0xD0000141	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_ECC_PUBLIC_VALUE_Y	0xD0000241	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_ECC_PRIVATE_VALUE	0xC0000341	Protected	Ref	bignum	d
QCOM_CRYPT0_ATTR_ECC_CURVE	0xF0000441	Public	Value	int	Value from Table 7-18
QCOM_CRYPT0_ATTR_ED25519_PUBLIC_VALUE	0xD00001C0	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_ED25519_PRIVATE_VALUE	0xC00002C0	Protected	Ref	bignum	
QCOM_CRYPT0_ATTR_CURVE25519_PUBLIC_VALUE	0xD00001C1	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_CURVE25519_PRIVATE_VALUE	0xC00002C1	Protected	Ref	bignum	
QCOM_CRYPT0_ATTR_SRP_PRIME	0xD00001C2	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_SRP_GEN	0xD00002C2	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_SRP_VERIFIER	0xC00003C2	Protected	Ref	bignum	
QCOM_CRYPT0_ATTR_SRP_USERNAME	0xD00004C2	Public	Ref	string	
QCOM_CRYPT0_ATTR_SRP_PASSWORD	0xC00005C2	Protected	Ref	string	
QCOM_CRYPT0_ATTR_SRP_TYPE	0xD00006C2	Public	Value	int	
QCOM_CRYPT0_ATTR_SRP_HASH	0xD00007C2	Public	Value	int	
QCOM_CRYPT0_ATTR_SRP_SALT	0xD00008C2	Public	Ref	bignum	
QCOM_CRYPT0_ATTR_SRP_PUBLIC_VALUE	0xD00009C2	Public	Ref	bignum	

Name	Value	Protection	Type	Format (Table 7-20)	Comment
QCOM_CRYPTO_ATTR_SRP_PRIVATE_VALUE	0xC0000AC2	Protected	Ref	bignum	

Table 7-20 Attribute format definitions

Format	Description
binary	An array of unsigned octets
bignum	An unsigned bignum in big-endian binary format. Leading zero bytes are allowed.
int	Values attributes represented in a single integer returned/read from argumenta.

Additional attributes may be defined for use with implementation defined algorithms.

7.10 Miscellaneous APIs

7.10.1 qcom_sys_reset

Definition	Reset the system.	
Prototype	A_STATUS qcom_sys_reset (
)	
Return value	A_OK / A_ERROR	

7.10.2 qcom_mac_get

Definition	Get QCA4010/QCA4012 MAC address.	
Prototype	A_STATUS qcom_mac_get (
	A_UINT8 device_id,	Interface ID (default is 0)
	A_UINT8 *pmac,	A pointer to MAC address
)	
Return value	A_OK / A_ERROR	

7.10.3 qcom_timer_init

Definition	Initialize a timer.	
Prototype	int qcom_mac_get (
	qcom_timer_t* qtimer	A pointer to qcom_timer_t for saving internal data of timer.
	void (*fn)(unsigned int, void *),	Call back function for the timer
	void* arg,	The parameters of call back function
	int timeout,	Timeout value for the timer in milliseconds.
	qcom_timer_type_e type	Oneshot or Periodic.

)	
Return value	0 on success, else on error.	

7.10.4 qcom_timer_start

Definition	Start a timer.	
Prototype	int qcom_timer_start (
	<i>qcom_timer_t* qtimer</i>	A pointer to qcom_timer_t returned from qcom_timer_init.
)	
Return value	0 on success, else on error.	

7.10.5 qcom_timer_stop

Definition	Stop a timer.	
Prototype	int qcom_timer_stop (
	<i>qcom_timer_t* qtimer</i>	A pointer to qcom_timer_t returned from qcom_timer_init.
)	
Return value	0 on success, else on error.	

7.10.6 qcom_timer_delete

Definition	Delete a timer.	
Prototype	int qcom_timer_delete (
	<i>qcom_timer_t* qtimer</i>	A pointer to qcom_timer_t returned from qcom_timer_init.
)	
Return value	0 on success, else on error.	

7.10.7 qcom_timer_us_start

Definition	Start the timer work in μ s.	
Prototype	int qcom_timer_us_start(
	<i>qcom_timer_t * qtimer</i>	Timer configuration parameter
)	
Return value	0 on success, else on error.	

7.10.8 qcom_time_us

Definition	Get current time in μ s.	
Prototype	A_UINT64 qcom_time_us(
)	

Return value	Current time in μ s.
---------------------	--------------------------

7.10.9 qcom_watchdog

Definition	Enable/disable a watchdog timer.	
Prototype	int qcom_watchdog (
	<i>int enable</i> ,	APP_WDT_DISABLE – disable watchdog APP_WDT_ENABLE – enable watchdog
	<i>int timeout</i>	Watchdog timer expiry timeout in seconds
)	
Description	If the watchdog timer is enabled, it will be reset by an internal maintenance timer every 5 seconds. The developer can use the qcom_watchdog_feed() API to reset the timer from application, if desired.	
Return value	0	

7.10.10 qcom_watchdog_feed

Definition	Reset the watchdog timer.	
Prototype	void qcom_watchdog_feed (
)	
Return value	none	

7.10.11 qcom_suspend_restore_flag_get

Definition	Get the reset reason.	
Prototype	void qcom_suspend_restore_flag_get (
	<i>A_UINT8 *flg</i>	1 = Wake-up from software (timer) 2 = Wake-up from hardware (GPIO) 0 = Normal reset
)	
Return value	void	

7.10.12 qcom_enable_print

Definition	Enable/Disable console printf. This API must be called at the beginning of user application if UART is used.	
Prototype	void qcom_enable_print (
	<i>int enable</i>	1 = Enable console printf. 0 = Disable console printf.
)	
Return value	void	

7.10.13 qcom_vsnprintf

Definition	Format output of a variable argument list, write at most <i>n</i> bytes (including the terminating null byte ('\0')) to <i>buffer</i> .	
Prototype	Int qcom_vsnprintf (
	<i>buffer</i> ,	The buffer for formatting output.
	<i>size_t n</i>	Maximum number of characters to write.
	<i>const char *format</i>	Format specification.
	<i>va_list ap</i>	A list of arguments.
)	
Return value	If successful return, return the number of characters printed, otherwise, and return a negative value.	

7.10.14 qcom_console_get_free_txbuf_sz

Definition	Query the current available Tx ring buffer size	
Prototype	A_UINT32 qcom_console_get_free_txbuf_sz (
)	
Return value	The current available Tx ring buffer size in bytes.	

7.10.15 qcom_aes_encrypt_init

Definition	AES encryption initialize.	
Prototype	void * qcom_aes_encrypt_init (
	<i>unsigned char *key</i> ,	The key used in AES encryption algorithm
	<i>int len</i>	The length in bytes of the key. Must be 16
)	
Return value	The pointer of AES context, NULL on error.	

7.10.16 qcom_aes_encrypt

Definition	AES encryption operation.	
Prototype	void qcom_aes_encrypt (
	<i>void *ctx</i> ,	The context of AES encryption
	<i>unsigned char *plain</i> ,	The input plain data to be encrypted
	<i>unsigned char *crypt</i>	The output cipher data
)	
Return value	None.	

7.10.17 qcom_aes_encrypt_deinit

Definition	Release AES context.	
Prototype	void qcom_aes_encrypt_deinit (
	<i>void *ctx</i>	The context of AES encryption
)	

Return value	None.
---------------------	-------

7.10.18 qcom_aes_decrypt_init

Definition	AES decryption initialize.	
Prototype	void *qcom_aes_decrypt_init (
	<i>unsigned char *key,</i>	The key used in AES decryption algorithm
	<i>int len</i>	The length in bytes of the key. Must be 16
)	
Return value	The pointer of AES context, NULL on error.	

7.10.19 qcom_aes_decrypt

Definition	AES decryption operation.	
Prototype	void qcom_aes_decrypt (
	<i>void *ctx,</i>	The context of AES decryption
	<i>unsigned char *crypt,</i>	The input cipher data to be decrypted
	<i>unsigned char *plain</i>	The output plain data
)	
Return value	None.	

7.10.20 qcom_aes_decrypt_deinit

Definition	Release AES context.	
Prototype	void qcom_aes_decrypt_deinit (
	<i>void *ctx</i>	The context of AES decryption
)	
Return value	None.	

7.10.21 qcom_time_us

Definition	Return A_UINT64 type value with microsecond.	
Prototype	qcom_time_us (
)	
Description	The API returns A_UINT64 type value. It reads the system low frequency timer register and returns the time value (unit in microsecond). The clock of the timer is 32768 Hz indicating that the accuracy of the time is about 30.5 microseconds.	
Return value	A_UINT64	

7.10.22 qcom_isr_handler_install

Definition	For callback function register use.	
Prototype	A_UINT32 qcom_isr_handler_install (

	<i>qcom_gpio_isr_info_t *gpio_isr</i>	ISR type information
)	
Description	This is for callback function register use. To run cb function in a higher priority, register it by <code>qcom_isr_handler_install()</code> . If normal interrupt function will be used, <code>qcom_gpio_interrupt_register</code> must also be registered.	
Return value	A_UINT32	

7.10.23 qcom_cust_speed_ctrl

Definition	Set user-defined action of dedicated GPIO pin.	
Prototype	A_UINT32 <code>qcom_cust_speed_ctrl(</code>	
	<i>A_UINT32 pin</i>	GPIO pin Choose GPIO pins as output control.
	<i>A_UINT32 us</i>	microsecond Delay microsecond for no action and maintain the original state.
	<i>A_UINT32 ms</i>	millisecond Pull GPIO as output high for millisecond. Minimum is 2 ms. In the end, it pulls low GPIO.
)	
Description	Before using it, <code>qcom_hf_timer_install()</code> must be called to install the HF timer interrupt. It chooses one GPIO pin as the control pin. For microsecond timer, this API uses high frequency timer to get a more accurate time value with 65 MHz clock. The API delays “us” microseconds with doing nothing and then pulls GPIO as output high for “ms” millisecond.	
Return value	A_UINT32	

A P2P Linux Client Commands

A.1 Operation groups overview

wpa_cli	wpa_cli			
Device Discovery	p2p_find	p2p_listen	p2p_stop_find	p2p_flush
Group Formation	p2p_prov_disc	p2p_connect	p2p_group_add	p2p_reject
	p2p_group_remove	p2p_cancel	p2p_remove_client	
Invitation	p2p_invite			
Group Operations	wps_pin	wps_pbc	p2p_get_passphrase	p2p_presence_req
Parameters	p2p_ext_listen	p2p_set	set	
Status	p2p_peers	p2p_peer		
Group Status	status	sta	all_sta	list_network
	remove_network			

A.2 wpa_cli

Actual Wi-Fi P2P operations are requested during runtime.

These can be done for example using wpa_cli (which is described below) or a GUI like wpa_gui-qt4.

wpa_cli starts in interactive mode if no command string is included on the command line. By default, it selects the first network interface that it can find (and that wpa_supplicant controls). If more than one interface is in use, it may be necessary to select one of the explicitly by adding -i argument on the command line (for example, 'wpa_cli -i wlan1'). Most of the P2P operations are done on the main interface (for example, the interface, such as wlan0, that is automatically added when the driver is loaded). When using a separate virtual interface for group operations (for example, wlan1), the control interface for that group interface may need to be used for some operations (mainly WPS activation in GO). This can change in the future so that all the needed operations could be done over the main control interface.

A.3 Device discovery

A.3.1 p2p_find

```
[timeout in seconds] [type=<social|progressive>]
[dev_id=<addr>] [dev_type=<device type>]
[delay=<search delay in ms>]
```

The default behavior is to run a single full scan in the beginning and then scan only social channels. `type=social` will scan only social channels (that is, it skips the initial full scan). `type=progressive` is like the default behavior, but it will scan through all the channels progressively one channel at the time in the Search state rounds. This will help in finding new groups or groups missed during the initial full scan.

The optional `dev_id` option can be used to specify a single P2P peer to search for. The optional `delay` parameter can be used to request an extra delay to be used between search iterations (for example, to free up radio resources for concurrent operations).

The optional `dev_type` option can be used to specify a single device type (primary or secondary) to search for (for example, "`p2p_find dev_type=1-0050F204-1`").

A.3.2 p2p_listen

[timeout in seconds]

Start Listen-only state (become discoverable without searching for other devices). Optional parameter can be used to specify the duration for the Listen operation in seconds. This command may not be of that much use during normal operations and is mainly designed for testing. It can also be used to keep the device discoverable without having to maintain a group.

A.3.3 p2p_stop_find

Stop ongoing P2P device discovery or other operation (listen mode).

A.3.4 p2p_flush

Flush P2P peer table and state.

A.4 Group formation

A.4.1 p2p_prov_disc

<peer device address> <display|keypad|pbc> [join|auto]

Send P2P provision discovery request to the specified peer. The parameters for this command are the P2P device address of the peer and the desired configuration method. For example, "`p2p_prov_disc 02:01:02:03:04:05 display`" request the peer to display a PIN for us.

"`p2p_prov_disc 02:01:02:03:04:05 keypad`" request the peer to enter a PIN that we display.

The optional "join" parameter can be used to indicate that this command is requesting an already running GO to prepare for a new client.

This is mainly used with "display" to request it to display a PIN. The "auto" parameter can be used to request `wpa_supplicant` to automatically figure out whether the peer device is operating as a GO and if so, use join-a-group style PD instead of GO Negotiation style PD.

A.4.2 p2p_connect

```
<peer device address> <pb|pin|PIN#> [display|keypad]
[persistent|persistent=<network id>] [join|auth]
[go_intent=<0..15>] [freq=<in MHz>] [ht40] [vht] [provdisc]
```

Start P2P group formation with a discovered P2P peer.

This includes:

optional group owner negotiation, group interface setup, provisioning, and establishing data connection.

The <pb|pin|PIN#> parameter specifies the WPS provisioning method. "pb" string starts pushbutton method, "pin" string start PIN method using an automatically generated PIN (which will be returned as the command return code), PIN# means that a preselected PIN can be used (for example, 12345670).

[display|keypad] is used with PIN method to specify which PIN is used (display=dynamically generated random PIN from local display, keypad=PIN entered from peer display).

"persistent" parameter can be used to request a persistent group to be formed. The "persistent=<network id>" alternative can be used to pre-populate SSID/passphrase configuration based on a previously used persistent group where this device was the GO. The previously used parameters will then be used if the local end becomes the GO in GO Negotiation (which can be forced with go_intent=15).

"join" indicates that this is a command to join an existing group as a client. It skips the GO Negotiation part. This will send a Provision Discovery Request message to the target GO before associating for WPS provisioning.

"auth" indicates that the WPS parameters are authorized for the peer device without actually starting GO Negotiation (that is, the peer is expected to initiate GO Negotiation). This is mainly for testing purposes.

"go_intent" can be used to override the default GO Intent for this GO Negotiation.

"freq" can be used to set a forced operating channel (for example, freq=2412 to select 2.4 GHz channel 1).

"provdisc" can be used to request a Provision Discovery exchange to be used prior to starting GO Negotiation as a workaround with some deployed P2P implementations that require this to allow the user to accept the connection.

A.4.3 p2p_group_add

```
[persistent|persistent=<network id>] [freq=<freq in MHz>]ht40] [vht]
```

Set up a P2P group owner manually (that is, without group owner negotiation with a specific peer). This is also known as autonomous GO. Optional persistent=<network id> can be used to specify restart of a persistent group. Optional freq=<freq in MHz> can be used to force the GO to be started on a specific frequency. Special freq=2 or freq=5 options can be used to request the best 2.4 GHz or 5 GHz band channel to be selected automatically.

A.4.4 p2p_reject

```
<peer device address>
```

Reject connection attempt from a peer (specified with a device address). This is a mechanism to reject a pending GO Negotiation with a peer and request to automatically block any further connection or discovery of the peer.

A.4.5 p2p_group_remove

<group interface>

Terminate a P2P group. If a new virtual network interface was used for the group, it will also be removed. The network interface name of the group interface is used as a parameter for this command.

A.4.6 p2p_cancel

Cancel an ongoing P2P group formation and joining-a-group related operation. This operation unauthorizes the specific peer device (if any had been authorized to start group formation), stops P2P find (if in progress), stops pending operations for join-a-group, and removes the P2P group interface (if one was used) that is in the WPS provisioning step. If the WPS provisioning step has been completed, the group is not terminated.

A.4.7 p2p_remove_client

<peer's P2P Device Address|iface=<interface address>

This command can be used to remove the specified client from all groups (operating and persistent) from the local GO. Note that the peer device can rejoin the group if it is in possession of a valid key. See p2p_set per_sta_psk command below for more details on how the peer can be removed securely.

A.5 Invitation

A.5.1 p2p_invite

```
[persistent=<network id>|group=<group ifname>] [peer=address]
[go_dev_addr=address] [freq=<freq in MHz>] [ht40] [vht]
[pref=<MHz>]
```

Invite a peer to join a group (for example, group=wlan1) or to re-invoke a persistent group (for example, persistent=4). If the peer device is the GO of the persistent group, the peer parameter is not needed. Otherwise it is used to specify which device to invite. The go_dev_addr parameter can be used to override the GO device address for Invitation Request should it be not known for some reason (this should not be needed in most cases). When re-invoking a persistent group, the GO device can specify the frequency for the group with the freq parameter. When re-invoking a persistent group, the P2P client device can use freq parameter to force a specific operating channel (or invitation failure if GO rejects that) or pref parameter to request a specific channel (while allowing GO to select to use another channel, if needed).

A.6 Group operations

These are used on the group interface.

A.6.1 wps_pin

```
<any|address> <PIN>
```

Start WPS PIN method. This allows a single WPS Enrollee to connect to the AP/GO. This is used on the GO when a P2P client joins an existing group. The second parameter is the address of the Enrollee or a string "any" to allow any station to use the entered PIN (which will restrict the PIN for one-time-use). PIN is the Enrollee PIN read either from a label or display on the P2P Client/WPS Enrollee.

A.6.2 wps_pbc

Start WPS PBC method (that is, push the button). This allows a single WPS Enrollee to connect to the AP/GO. This is used on the GO when a P2P client joins an existing group.

A.6.3 p2p_get_passphrase

Get the passphrase for a group (only available when acting as a GO).

A.6.4 p2p_presence_req

```
[<duration> <interval>] [<duration> <interval>]
```

Send a P2P Presence Request to the GO (this is only available when acting as a P2P client). If no duration/interval pairs are given, the request indicates that this client has no special needs for GO presence. The first parameter pair gives the preferred duration and interval values in microseconds. If the second pair is included, that indicates which value would be acceptable. This command returns OK immediately and the response from the GO is indicated in a P2P-PRESENCE-RESPONSE event message.

A.7 Parameters

A.7.1 p2p_ext_listen

```
[<period> <interval>]
```

Configure Extended Listen Timing. If the parameters are omitted, this feature is disabled. If the parameters are included, Listen State will be entered every interval msec for at least period msec. Both values have acceptable range of 1-65535 (with interval obviously having to be larger than or equal to duration). If the P2P module is not idle at the time the Extended Listen Timing timeout occurs, the Listen State operation will be skipped. The configured values will also be advertised to other P2P Devices. The received values are available in the p2p_peer command output:

```
ext_listen_period=100 ext_listen_interval=5000
```

A.7.2 p2p_set

<field> <value>

Change dynamic P2P parameters.

p2p_set discoverability <0/1>

Disable/enable advertisement of client discoverability. This is enabled by default and this parameter is mainly used to allow testing of device discoverability.

p2p_set managed <0/1>

Disable/enable managed P2P Device operations. This is disabled by default.

p2p_set listen_channel <1/6/11>

Set P2P Listen channel. This is mainly meant for testing purposes and changing the Listen channel during normal operations can result in protocol failures.

p2p_set ssid_postfix <postfix>

Set postfix string to be added to the automatically generated P2P SSID (DIRECT-<two random characters>). For example, postfix of "-testing" Could result in the SSID becoming DIRECT-ab-testing.

p2p_set per_sta_psk <0/1>

Disabled (default)/enables use of per-client PSK in the P2P groups. This can be used to request GO to assign a unique PSK for each client during WPS provisioning. When enabled, this allow clients to be removed from the group securely with p2p_remove_client command since that client's PSK is removed at the same time to prevent it from connecting back using the old PSK. When per-client PSK is not used, the client can still be disconnected, but it will be able to re-join the group since the PSK it learned previously is still valid. It should be noted that the default passphrase on the GO that is normally used to allow legacy stations to connect through manual configuration does not change here, so if that is shared, devices with knowledge of that passphrase can still connect.

A.7.3 set

<field> <value>

Set global configuration parameters which may also affect P2P operations. The format on these parameters is same as is used in wpa_supplicant.conf. Only the parameters listen here should be changed. Modifying other parameters may result in incorrect behavior since not all existing users of the parameters are updated.

set uuid <UUID>

Set WPS UUID (by default, this is generated based on the MAC address).

set device_name <device name>

Set WPS Device Name (also included in some P2P messages).

set manufacturer <manufacturer>

Set WPS Manufacturer.

set model_name <model name>

Set WPS Model Name.

set model_number <model number>

Set WPS Model Number.

```
set serial_number <serial number>
```

Set WPS Serial Number.

```
set device_type <device type>
```

Set WPS Device Type.

```
set os_version <OS version>
```

Set WPS OS Version.

```
set config_methods <config methods>
```

Set WPS Configuration Methods.

```
set sec_device_type <device type>
```

Add a new Secondary Device Type.

```
set p2p_go_intent <GO intent>
```

Set the default P2P GO Intent.

NOTE: This value can be overridden in `p2p_connect` command and as such, there should be no need to change the default value here during normal operations.

```
set p2p_ssid_postfix <P2P SSID postfix>
```

Set P2P SSID postfix.

```
set persistent_reconnect <0/1>
```

Disable/enabled persistent reconnect for re-invocation of persistent groups. If enabled, invitations to re-invoke a persistent group will be accepted without separate authorization (for example, user interaction).

A.8 Status

A.8.1 p2p_peers

[discovered]

List P2P Device Addresses of all the P2P peers we know. The optional "discovered" parameter filters out the peers that we have not fully discovered, that is, which we have only seen in a received Probe Request frame.

A.8.2 p2p_peer

<P2P Device Address>

Fetch information about a known P2P peer.

A.9 Group status

These are used on the group interface.

A.9.1 status

Show status information (connection state, role, use encryption parameters, IP address, etc.).

A.9.2 sta

Show information about an associated station (when acting in AP/GO role).

A.9.3 all_sta

Lists the currently associated stations. Configuration data.

A.9.4 list_network

Lists the configured networks, including stored information for persistent groups. The identifier in this list is used with `p2p_group_add` and `p2p_invite` to indicate which persistent group is to be re-invoked.

A.9.5 remove_network

`<network id>`

Remove a network entry from configuration.