

OPL1000_WIFI_BLE_API_GUIDE

1.0.1.25

Generated by Doxygen 1.8.14

Contents

1	SDK PREVIEW	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	Module Documentation	9
4.1	BLE ALL APIs	9
4.1.1	Detailed Description	9
4.2	BLE CM APIs	10
4.2.1	Detailed Description	11
4.2.2	Typedef Documentation	11
4.2.2.1	LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM_T	11
4.2.2.2	LE_CM_MSG_ADD_TO_WHITE_LIST_CFM_T	11
4.2.2.3	LE_CM_MSG_CANCEL_CONNECTION_CFM_T	12
4.2.2.4	LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM_T	12
4.2.2.5	LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T	12
4.2.2.6	LE_CM_MSG_CREATE_CONNECTION_CFM_T	12
4.2.2.7	LE_CM_MSG_ENTER_ADVERTISING_CFM_T	12
4.2.2.8	LE_CM_MSG_ENTER_SCANNING_CFM_T	12
4.2.2.9	LE_CM_MSG_EXIT_ADVERTISING_CFM_T	12
4.2.2.10	LE_CM_MSG_EXIT_SCANNING_CFM_T	12
4.2.2.11	LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T	13

4.2.2.12	LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T	13
4.2.2.13	LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T	13
4.2.2.14	LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T	13
4.2.2.15	LE_CM_MSG_SET_CHANNEL_MAP_CFM_T	13
4.2.2.16	LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T	13
4.2.2.17	LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T	13
4.2.2.18	LE_CM_MSG_SET_SCAN_PARAMS_CFM_T	13
4.2.2.19	LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T	14
4.2.3	Enumeration Type Documentation	14
4.2.3.1	anonymous enum	14
4.2.4	Function Documentation	15
4.2.4.1	LeCmInit()	15
4.3	BLE GAP APIs	16
4.3.1	Detailed Description	18
4.3.2	Macro Definition Documentation	18
4.3.2.1	GAP_ADTYPE_128BIT_COMPLETE	18
4.3.2.2	GAP_ADTYPE_128BIT_MORE	18
4.3.2.3	GAP_ADTYPE_16BIT_COMPLETE	18
4.3.2.4	GAP_ADTYPE_16BIT_MORE	19
4.3.2.5	GAP_ADTYPE_32BIT_COMPLETE	19
4.3.2.6	GAP_ADTYPE_32BIT_MORE	19
4.3.2.7	GAP_ADTYPE_3D_INFO_DATA	19
4.3.2.8	GAP_ADTYPE_ADV_INTERVAL	19
4.3.2.9	GAP_ADTYPE_APPEARANCE	19
4.3.2.10	GAP_ADTYPE_FLAGS	19
4.3.2.11	GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED	19
4.3.2.12	GAP_ADTYPE_FLAGS_GENERAL	20
4.3.2.13	GAP_ADTYPE_FLAGS_LIMITED	20
4.3.2.14	GAP_ADTYPE_LE_BD_ADDR	20
4.3.2.15	GAP_ADTYPE_LE_ROLE	20

4.3.2.16	GAP_ADTYPE_LOCAL_NAME_COMPLETE	20
4.3.2.17	GAP_ADTYPE_LOCAL_NAME_SHORT	20
4.3.2.18	GAP_ADTYPE_MANUFACTURER_SPECIFIC	20
4.3.2.19	GAP_ADTYPE_OOB_CLASS_OF_DEVICE	20
4.3.2.20	GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC	21
4.3.2.21	GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDR	21
4.3.2.22	GAP_ADTYPE_POWER_LEVEL	21
4.3.2.23	GAP_ADTYPE_PUBLIC_TARGET_ADDR	21
4.3.2.24	GAP_ADTYPE_RANDOM_TARGET_ADDR	21
4.3.2.25	GAP_ADTYPE_SERVICE_DATA	21
4.3.2.26	GAP_ADTYPE_SERVICE_DATA_128BIT	21
4.3.2.27	GAP_ADTYPE_SERVICE_DATA_32BIT	21
4.3.2.28	GAP_ADTYPE_SERVICES_LIST_128BIT	22
4.3.2.29	GAP_ADTYPE_SERVICES_LIST_16BIT	22
4.3.2.30	GAP_ADTYPE_SIGNED_DATA	22
4.3.2.31	GAP_ADTYPE_SIMPLE_PAIRING_HASHC_256	22
4.3.2.32	GAP_ADTYPE_SIMPLE_PAIRING_RANDR_256	22
4.3.2.33	GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE	22
4.3.2.34	GAP_ADTYPE_SM_OOB_FLAG	22
4.3.2.35	GAP_ADTYPE_SM_TK	22
4.3.2.36	GAP_PUBLIC_ADDR	23
4.3.2.37	GAP_RAND_ADDR_NRPA	23
4.3.2.38	GAP_RAND_ADDR_RPA	23
4.3.2.39	GAP_RAND_ADDR_STATIC	23
4.3.2.40	GAP_SCAN_TYPE_ACTIVE	23
4.3.2.41	GAP_SCAN_TYPE_PASSIVE	23
4.3.2.42	GAP_TX_PWR_CURR_VAL	23
4.3.2.43	GAP_TX_PWR_MAX_VAL	23
4.3.2.44	GAPBOND_IO_CAP_DISPLAY_ONLY	24
4.3.2.45	GAPBOND_IO_CAP_DISPLAY_YES_NO	24

4.3.2.46	GAPBOND_IO_CAP_KEYBOARD_DISPLAY	24
4.3.2.47	GAPBOND_IO_CAP_KEYBOARD_ONLY	24
4.3.2.48	GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT	24
4.3.2.49	GAPBOND_PAIRING_MODE_INITIATE	24
4.3.2.50	GAPBOND_PAIRING_MODE_NO_PAIRING	24
4.3.2.51	GAPBOND_PAIRING_MODE_WAIT_FOR_REQ	24
4.3.2.52	LE_GAP_ADV_MAX_SIZE	25
4.3.3	Function Documentation	25
4.3.3.1	LeGapAddToResolvingList()	25
4.3.3.2	LeGapAddToWhiteList()	25
4.3.3.3	LeGapAdvertisingEnable()	26
4.3.3.4	LeGapCentralConnectReq()	26
4.3.3.5	LeGapCentralSetDataChannel()	26
4.3.3.6	LeGapClearResolvingList()	27
4.3.3.7	LeGapClearWhiteList()	27
4.3.3.8	LeGapConnectCancelReq()	27
4.3.3.9	LeGapConnParaRequestRsp()	27
4.3.3.10	LeGapConnUpdateRequest()	28
4.3.3.11	LeGapConnUpdateResponse()	28
4.3.3.12	LeGapDisconnectReq()	29
4.3.3.13	LeGapGenRandAddr()	29
4.3.3.14	LeGapGetBtAddr()	29
4.3.3.15	LeGapReadAdvChannelTxPower()	30
4.3.3.16	LeGapReadChannelMap()	30
4.3.3.17	LeGapReadResolvingListSize()	30
4.3.3.18	LeGapReadRssi()	30
4.3.3.19	LeGapReadTxPower()	31
4.3.3.20	LeGapReadWhiteListSize()	31
4.3.3.21	LeGapRemoveFromWhiteList()	31
4.3.3.22	LeGapScanningReq()	32

4.3.3.23	LeGapSetAdvData()	32
4.3.3.24	LeGapSetAdvParameter()	33
4.3.3.25	LeGapSetConnParameter()	33
4.3.3.26	LeGapSetDataChannelPduLen()	33
4.3.3.27	LeGapSetRandAddr()	34
4.3.3.28	LeGapSetRpaTimeout()	34
4.3.3.29	LeGapSetStaticAddr()	35
4.3.3.30	LeSetScanParameter()	35
4.3.3.31	LeSetScanRspData()	35
4.4	BLE GATT APIs	37
4.4.1	Detailed Description	41
4.4.2	Macro Definition Documentation	41
4.4.2.1	CHAR_AGGREGATE_DESCRIPTOR	41
4.4.2.2	CHAR_CLIENT_CONFIG_DESCRIPTOR	42
4.4.2.3	CHAR_DECL_UUID16_ATTR_VAL	42
4.4.2.4	CHAR_EXT_PROP_DESCRIPTOR	42
4.4.2.5	CHAR_PRESENT_FORMAT_DESCRIPTOR	42
4.4.2.6	CHAR_SERVER_CONFIG_DESCRIPTOR	42
4.4.2.7	CHAR_USER_DESC_DESCRIPTOR	42
4.4.2.8	CHARACTERISTIC_DECL_UUID128	43
4.4.2.9	CHARACTERISTIC_DECL_UUID16	43
4.4.2.10	CHARACTERISTIC_UUID128	43
4.4.2.11	CHARACTERISTIC_UUID16	43
4.4.2.12	GATT_CHAR_AGG_FORMAT_UUID	43
4.4.2.13	GATT_CHAR_EXT_PROPS_UUID	43
4.4.2.14	GATT_CHAR_FORMAT_UUID	44
4.4.2.15	GATT_CHAR_USER_DESC_UUID	44
4.4.2.16	GATT_CHARACTERISTIC_UUID	44
4.4.2.17	GATT_CLIENT_CHAR_CFG_UUID	44
4.4.2.18	GATT_EXT_REPORT_REF_UUID	44

4.4.2.19	GATT_INCLUDE_UUID	44
4.4.2.20	GATT_PRIMARY_SERVICE_UUID	44
4.4.2.21	GATT_REPORT_REF_UUID	44
4.4.2.22	GATT_SECONDARY_SERVICE_UUID	45
4.4.2.23	GATT_SERV_CHAR_CFG_UUID	45
4.4.2.24	GATT_VALID_RANGE_UUID	45
4.4.2.25	INCLUDE_DECL_UUID128	45
4.4.2.26	INCLUDE_DECL_UUID128_ATTR_VAL	45
4.4.2.27	INCLUDE_DECL_UUID16_ATTR_VAL	45
4.4.2.28	INCLUDE_DECL_UUINT16	45
4.4.2.29	LE_ATT_UUID_SIZE	46
4.4.2.30	LE_GATT_CHAR_PROP_AUTH	46
4.4.2.31	LE_GATT_CHAR_PROP_BCAST	46
4.4.2.32	LE_GATT_CHAR_PROP_EXT_PROP	46
4.4.2.33	LE_GATT_CHAR_PROP_IND	46
4.4.2.34	LE_GATT_CHAR_PROP_NTF	46
4.4.2.35	LE_GATT_CHAR_PROP_RD	46
4.4.2.36	LE_GATT_CHAR_PROP_WR	47
4.4.2.37	LE_GATT_CHAR_PROP_WR_NO_RESP	47
4.4.2.38	LE_GATT_CLIENT_CFG_INDICATION	47
4.4.2.39	LE_GATT_CLIENT_CFG_NOTIFICATION	47
4.4.2.40	LE_GATT_EXT_PROP_RELIABLE_WR	47
4.4.2.41	LE_GATT_EXT_PROP_WR_AUX	47
4.4.2.42	LE_GATT_FLAG_PREPARE_WRITE	47
4.4.2.43	LE_GATT_FLAG_WRITE_CMD	47
4.4.2.44	LE_GATT_FLAG_WRITE_REQ	48
4.4.2.45	LE_GATT_PERM_AUTH_READABLE	48
4.4.2.46	LE_GATT_PERM_AUTH_WRITABLE	48
4.4.2.47	LE_GATT_PERM_NONE	48
4.4.2.48	LE_GATT_PERM_READ	48

4.4.2.49	LE_GATT_PERM_RELIABLE_WRITE	48
4.4.2.50	LE_GATT_PERM_WRITE_CMD	48
4.4.2.51	LE_GATT_PERM_WRITE_REQ	48
4.4.2.52	LE_GATT_PERMIT_AUTHEN_READ	49
4.4.2.53	LE_GATT_PERMIT_AUTHEN_WRITE	49
4.4.2.54	LE_GATT_PERMIT_AUTHOR_READ	49
4.4.2.55	LE_GATT_PERMIT_AUTHOR_WRITE	49
4.4.2.56	LE_GATT_PERMIT_ENCRYPT_READ	49
4.4.2.57	LE_GATT_PERMIT_ENCRYPT_WRITE	49
4.4.2.58	LE_GATT_PERMIT_READ	49
4.4.2.59	LE_GATT_PERMIT_READABLE	49
4.4.2.60	LE_GATT_PERMIT_SC_AUTHEN_READ	50
4.4.2.61	LE_GATT_PERMIT_SC_AUTHEN_WRITE	50
4.4.2.62	LE_GATT_PERMIT_WRITABLE	50
4.4.2.63	LE_GATT_PERMIT_WRITE	50
4.4.2.64	PRIMARY_SERVICE_DECL_UUID128	50
4.4.2.65	PRIMARY_SERVICE_DECL_UUID16	50
4.4.2.66	SECONDARY_SERVICE_DECL_UUID128	50
4.4.2.67	SECONDARY_SERVICE_DECL_UUID16	51
4.4.3	Enumeration Type Documentation	51
4.4.3.1	anonymous enum	51
4.4.4	Function Documentation	52
4.4.4.1	LeGattAccessReadRsp()	52
4.4.4.2	LeGattAccessWriteRsp()	52
4.4.4.3	LeGattChangeAttrVal()	53
4.4.4.4	LeGattCharValConfirmation()	53
4.4.4.5	LeGattCharValIndicate()	54
4.4.4.6	LeGattCharValNotify()	54
4.4.4.7	LeGattExchangeMtuReq()	55
4.4.4.8	LeGattExchangeMtuRsp()	55

4.4.4.9	LeGattExecuteWriteCharValReliable()	56
4.4.4.10	LeGattFindAllCharacteristic()	56
4.4.4.11	LeGattFindAllCharDescriptor()	56
4.4.4.12	LeGattFindAllPrimaryService()	57
4.4.4.13	LeGattFindCharacteristicByUuid()	57
4.4.4.14	LeGattFindIncludedService()	58
4.4.4.15	LeGattFindPrimaryServiceByUuid()	58
4.4.4.16	LeGattGetAttrHandle()	59
4.4.4.17	LeGattGetAttrVal()	59
4.4.4.18	LeGattGetAttrValLen()	59
4.4.4.19	LeGattGetAttrValMaxLen()	61
4.4.4.20	LeGattInit()	61
4.4.4.21	LeGattModifyAttrVal()	62
4.4.4.22	LeGattPrepareWriteCharValReliable()	62
4.4.4.23	LeGattReadCharValByUuid()	63
4.4.4.24	LeGattReadCharValue()	63
4.4.4.25	LeGattReadLongCharVal()	64
4.4.4.26	LeGattReadMultipleCharVal()	64
4.4.4.27	LeGattRegisterIncludeService()	64
4.4.4.28	LeGattRegisterService()	65
4.4.4.29	LeGattSignedWriteNoRsp()	65
4.4.4.30	LeGattStopCurrentProcedure()	66
4.4.4.31	LeGattWriteCharVal()	66
4.4.4.32	LeGattWriteCharValReliable()	67
4.4.4.33	LeGattWriteLongCharVal()	67
4.4.4.34	LeGattWriteNoRsp()	68
4.4.5	Variable Documentation	68
4.4.5.1	gcCharacteristicUuid	68
4.4.5.2	gcCharAggregateUuid	68
4.4.5.3	gcCharExtPropUuid	69

4.4.5.4	gcCharFormatUuid	69
4.4.5.5	gcCharUserDescUuid	69
4.4.5.6	gcClientCharConfigUuid	69
4.4.5.7	gcExtReportRefUuid	69
4.4.5.8	gcIncludeUuid	69
4.4.5.9	gcPrimaryServiceUuid	69
4.4.5.10	gcReportRefUuid	69
4.4.5.11	gcSecondaryServiceUuid	70
4.4.5.12	gcServerCharConfigUuid	70
4.4.5.13	gcValidRangeUuid	70
4.5	BLE MSG APIs	71
4.5.1	Detailed Description	72
4.5.2	Macro Definition Documentation	72
4.5.2.1	LE_ATT_MSG_BASE	72
4.5.2.2	LE_CM_MSG_BASE	72
4.5.2.3	LE_GATT_MSG_BASE	73
4.5.2.4	LE_HCI_MSG_BASE	73
4.5.2.5	LE_L2CAP_MSG_BASE	73
4.5.2.6	LE_SMP_MSG_BASE	73
4.5.2.7	LE_SYS_MSG_BASE	73
4.5.2.8	MESSAGE_ALLOCATE	73
4.5.2.9	MESSAGE_BULID	73
4.5.2.10	MESSAGE_DATA_BULID	74
4.5.2.11	MESSAGE_OFFSET	74
4.5.2.12	T_HOUR	74
4.5.2.13	T_MIN	74
4.5.2.14	T_SEC	74
4.5.3	Typedef Documentation	74
4.5.3.1	MESSAGE	74
4.5.3.2	MESSAGEID	75

4.5.3.3	MsgData	75
4.5.3.4	MsgLock	75
4.5.3.5	MSGLOCK	75
4.5.3.6	MSGSUBID	75
4.5.3.7	MSGTIMER	75
4.5.3.8	Task	75
4.5.3.9	TASK	75
4.5.3.10	TASKHANDLER	76
4.5.3.11	TASKPACK	76
4.5.4	Enumeration Type Documentation	76
4.5.4.1	anonymous enum	76
4.5.5	Function Documentation	76
4.5.5.1	LeCancelAllMessage()	76
4.5.5.2	LeCancelAllSubMessage()	77
4.5.5.3	LeCancelFirstMessage()	77
4.5.5.4	LeCancelFirstSubMessage()	78
4.5.5.5	LeGetSubMsgId()	78
4.5.5.6	LeHostCreateTask()	78
4.5.5.7	LeHostMessageLoop()	79
4.5.5.8	LeSendMessage()	79
4.5.5.9	LeSendMessageAfter()	79
4.5.5.10	LeSendMessageUnlock()	80
4.5.5.11	LeSendSubMessage()	80
4.5.5.12	LeSendSubMessageAfter()	81
4.5.5.13	LeSendSubMessageUnlock()	81
4.6	BLE SMP APIs	83
4.6.1	Detailed Description	84
4.6.2	Macro Definition Documentation	84
4.6.2.1	LE_MAX_BOND_COUNT	84
4.6.2.2	LE_SM_IO_CAP_DISP_ONLY	84

4.6.2.3	LE_SM_IO_CAP_DISP_YES_NO	84
4.6.2.4	LE_SM_IO_CAP_KEYBOARD_DISP	85
4.6.2.5	LE_SM_IO_CAP_KEYBOARD_ONLY	85
4.6.2.6	LE_SM_IO_CAP_NO_IO	85
4.6.2.7	LE_SM_PAIR_MITM_NO	85
4.6.2.8	LE_SM_PAIR_MITM_YES	85
4.6.2.9	LE_SM_PAIR_OOB_NO	85
4.6.2.10	LE_SM_PAIR_OOB_YES	85
4.6.2.11	LE_SM_PAIR_SC_NO	85
4.6.2.12	LE_SM_PAIR_SC_YES	86
4.6.3	Enumeration Type Documentation	86
4.6.3.1	anonymous enum	86
4.6.3.2	anonymous enum	86
4.6.4	Function Documentation	87
4.6.4.1	LeSmpInit()	87
4.6.4.2	LeSmpOobAuthDataRsp()	87
4.6.4.3	LeSmpOobPresent()	87
4.6.4.4	LeSmpPasskeyInput()	88
4.6.4.5	LeSmpScOobComputeConfirmVal()	88
4.6.4.6	LeSmpScOobDataRsp()	89
4.6.4.7	LeSmpSecurityReq()	89
4.6.4.8	LeSmpSecurityRsp()	89
4.6.4.9	LeSmpSetDefaultConfig()	90
4.6.4.10	LeSmpUserConfirmRsp()	90
4.7	WIFI APIs	91
4.7.1	Detailed Description	92
4.7.2	Macro Definition Documentation	92
4.7.2.1	WIFI_BEACON_INTERVAL_LENGTH	92
4.7.2.2	WIFI_CAPABILITY_INFO_LENGTH	92
4.7.2.3	WIFI_LENGTH_802_11	92

4.7.2.4	WIFI_LENGTH_PASSPHRASE	93
4.7.2.5	WIFI_MAC_ADDRESS_LENGTH	93
4.7.2.6	WIFI_MAX_LENGTH_OF_SSID	93
4.7.2.7	WIFI_MAX_SCAN_AP_NUM	93
4.7.2.8	WIFI_MAX_SUPPORTED_RATES	93
4.7.3	Typedef Documentation	93
4.7.3.1	wifi_event_notify_cb_t	93
4.7.4	Function Documentation	94
4.7.4.1	wifi_event_process_handler()	94
4.7.4.2	wifi_install_default_event_handlers()	94
4.7.4.3	wifi_register_event_handler()	95
4.8	WIFI Common APIs	96
4.8.1	Detailed Description	96
4.8.2	Typedef Documentation	96
4.8.2.1	wifi_event_cb_t	96
4.8.3	Function Documentation	97
4.8.3.1	wifi_event_loop_init()	97
4.8.3.2	wifi_event_loop_send()	98
4.8.3.3	wifi_event_loop_set_cb()	98
4.8.3.4	wifi_event_process_handler()	99
4.9	WIFI STA APIs	100
4.9.1	Detailed Description	102
4.9.2	Typedef Documentation	102
4.9.2.1	wifi_event_handler_t	102
4.9.2.2	wifi_init_complete_cb_t	102
4.9.2.3	wifi_result_t	103
4.9.3	Function Documentation	103
4.9.3.1	wifi_auto_connect_del_ap_info()	103
4.9.3.2	wifi_auto_connect_get_ap_info()	103
4.9.3.3	wifi_auto_connect_get_ap_num()	104

4.9.3.4	wifi_auto_connect_get_mode()	104
4.9.3.5	wifi_auto_connect_init()	104
4.9.3.6	wifi_auto_connect_reset()	105
4.9.3.7	wifi_auto_connect_set_ap_num()	105
4.9.3.8	wifi_auto_connect_set_mode()	105
4.9.3.9	wifi_auto_connect_start()	106
4.9.3.10	wifi_config_get_bandwidth()	106
4.9.3.11	wifi_config_get_bssid()	107
4.9.3.12	wifi_config_get_channel()	107
4.9.3.13	wifi_config_get_dtim_interval()	107
4.9.3.14	wifi_config_get_listen_interval()	108
4.9.3.15	wifi_config_get_mac_address()	108
4.9.3.16	wifi_config_get_opmode()	109
4.9.3.17	wifi_config_get_skip_dtim()	109
4.9.3.18	wifi_config_get_ssid()	109
4.9.3.19	wifi_config_set_bandwidth()	110
4.9.3.20	wifi_config_set_bssid()	110
4.9.3.21	wifi_config_set_channel()	111
4.9.3.22	wifi_config_set_dtim_interval()	111
4.9.3.23	wifi_config_set_listen_interval()	112
4.9.3.24	wifi_config_set_mac_address()	112
4.9.3.25	wifi_config_set_opmode()	113
4.9.3.26	wifi_config_set_skip_dtim()	113
4.9.3.27	wifi_config_set_ssid()	114
4.9.3.28	wifi_connection_connect()	115
4.9.3.29	wifi_connection_disconnect_ap()	115
4.9.3.30	wifi_connection_disconnect_sta()	116
4.9.3.31	wifi_connection_get_rssi()	116
4.9.3.32	wifi_connection_register_event_handler()	116
4.9.3.33	wifi_connection_scan_start()	117

4.9.3.34	wifi_connection_unregister_event_handler()	118
4.9.3.35	wifi_deinit()	118
4.9.3.36	wifi_fast_connect_get_mode()	118
4.9.3.37	wifi_fast_connect_set_mode()	119
4.9.3.38	wifi_fast_connect_start()	119
4.9.3.39	wifi_get_config()	119
4.9.3.40	wifi_init()	120
4.9.3.41	wifi_scan_get_ap_list()	120
4.9.3.42	wifi_scan_get_ap_num()	121
4.9.3.43	wifi_scan_get_ap_records()	121
4.9.3.44	wifi_scan_scan_stop()	122
4.9.3.45	wifi_scan_start()	122
4.9.3.46	wifi_set_config()	122
4.9.3.47	wifi_sta_get_ap_info()	123
4.9.3.48	wifi_start()	123
4.9.3.49	wifi_stop()	124
4.10	Enumeration	125
4.10.1	Detailed Description	125
4.10.2	Enumeration Type Documentation	125
4.10.2.1	wifi_auth_mode_t	125
4.10.2.2	wifi_bandwidth_t	126
4.10.2.3	wifi_cipher_type_t	126
4.10.2.4	wifi_event_t	126
4.10.2.5	wifi_mode_t	127
4.10.2.6	wifi_reason_code_t	127
4.10.2.7	wifi_scan_method_t	128
4.10.2.8	wifi_scan_type_t	129
4.10.2.9	wifi_sort_method_t	129

5	Data Structure Documentation	131
5.1	auto_conn_info_t Struct Reference	131
5.1.1	Field Documentation	131
5.1.1.1	ap_channel	131
5.1.1.2	beacon_interval	132
5.1.1.3	bssid	132
5.1.1.4	capabilities	132
5.1.1.5	dtim_prod	132
5.1.1.6	fast_connect	132
5.1.1.7	free_ocpy	132
5.1.1.8	hid_ssid	132
5.1.1.9	latest_beacon_rx_time	132
5.1.1.10	passphrase	133
5.1.1.11	psk	133
5.1.1.12	rsn_ie	133
5.1.1.13	rsni	133
5.1.1.14	ssid	133
5.1.1.15	supported_rates	133
5.1.1.16	wpa_data	133
5.1.1.17	wpa_ie	133
5.2	auto_connect_cfg_t Struct Reference	134
5.2.1	Field Documentation	134
5.2.1.1	flag	134
5.2.1.2	front	134
5.2.1.3	max_save_num	134
5.2.1.4	pFCInfo	134
5.2.1.5	rear	135
5.2.1.6	retryCount	135
5.2.1.7	targetIdx	135
5.2.1.8	uFCApNum	135

5.3	event_msg_t Struct Reference	135
5.3.1	Detailed Description	135
5.3.2	Field Documentation	135
5.3.2.1	event	136
5.3.2.2	length	136
5.3.2.3	param	136
5.4	hap_control_t Struct Reference	136
5.4.1	Field Documentation	136
5.4.1.1	hap_ap_info	136
5.4.1.2	hap_bitvector	136
5.4.1.3	hap_en	137
5.4.1.4	hap_final_index	137
5.4.1.5	hap_index	137
5.4.1.6	hap_ssid	137
5.5	LE_BT_ADDR_T Struct Reference	137
5.5.1	Field Documentation	137
5.5.1.1	addr	137
5.5.1.2	type	138
5.6	LE_CM_CONNECTION_COMPLETE_IND_T Struct Reference	138
5.6.1	Field Documentation	138
5.6.1.1	conn_hdl	138
5.6.1.2	conn_interval	138
5.6.1.3	conn_latency	138
5.6.1.4	dev_id	139
5.6.1.5	peer_addr	139
5.6.1.6	peer_addr_type	139
5.6.1.7	role	139
5.6.1.8	status	139
5.6.1.9	supervision_timeout	139
5.7	LE_CM_MSG_ADVERTISE_REPORT_IND_T Struct Reference	139

5.7.1	Field Documentation	140
5.7.1.1	addr	140
5.7.1.2	addr_type	140
5.7.1.3	data	140
5.7.1.4	event_type	140
5.7.1.5	len	140
5.7.1.6	rsi	140
5.8	LE_CM_MSG_CONN_PARA_REQ_T Struct Reference	140
5.8.1	Field Documentation	141
5.8.1.1	conn_hdl	141
5.8.1.2	itv_max	141
5.8.1.3	itv_min	141
5.8.1.4	latency	141
5.8.1.5	sv_tmo	141
5.9	LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T Struct Reference	141
5.9.1	Field Documentation	142
5.9.1.1	conn_hdl	142
5.9.1.2	interval	142
5.9.1.3	latency	142
5.9.1.4	status	142
5.9.1.5	supervision_timeout	142
5.10	LE_CM_MSG_DATA_LEN_CHANGE_IND_T Struct Reference	142
5.10.1	Field Documentation	143
5.10.1.1	conn_hdl	143
5.10.1.2	max_rx_octets	143
5.10.1.3	max_rx_time	143
5.10.1.4	max_tx_octets	143
5.10.1.5	max_tx_time	143
5.11	LE_CM_MSG_DIRECT_ADV_REPORT_IND_T Struct Reference	143
5.11.1	Field Documentation	144

5.11.1.1	direct_addr	144
5.11.1.2	direct_addr_type	144
5.11.1.3	peer_addr	144
5.11.1.4	peer_addr_type	144
5.11.1.5	rsi	144
5.12	LE_CM_MSG_DISCONNECT_COMPLETE_IND_T Struct Reference	144
5.12.1	Field Documentation	145
5.12.1.1	conn_hdl	145
5.12.1.2	reason	145
5.12.1.3	status	145
5.13	LE_CM_MSG_ENCRYPTION_CHANGE_IND_T Struct Reference	145
5.13.1	Field Documentation	145
5.13.1.1	conn_hdl	146
5.13.1.2	devid	146
5.13.1.3	enabled	146
5.13.1.4	status	146
5.14	LE_CM_MSG_ENCRYPTION_REFRESH_IND_T Struct Reference	146
5.14.1	Field Documentation	146
5.14.1.1	conn_hdl	146
5.14.1.2	devid	147
5.14.1.3	enabled	147
5.14.1.4	status	147
5.15	LE_CM_MSG_INIT_COMPLETE_CFM_T Struct Reference	147
5.15.1	Field Documentation	147
5.15.1.1	status	147
5.16	LE_CM_MSG_LTK_REQ_IND_T Struct Reference	147
5.16.1	Field Documentation	148
5.16.1.1	conn_hdl	148
5.16.1.2	devid	148
5.16.1.3	ediv	148

5.16.1.4	rand	148
5.17	LE_CM_MSG_READ_ADV_TX_POWER_CFM_T Struct Reference	148
5.17.1	Field Documentation	149
5.17.1.1	pwr_level	149
5.17.1.2	status	149
5.18	LE_CM_MSG_READ_BD_ADDR_CFM_T Struct Reference	149
5.18.1	Field Documentation	149
5.18.1.1	bd_addr	149
5.18.1.2	status	149
5.19	LE_CM_MSG_READ_CHANNEL_MAP_CFM_T Struct Reference	150
5.19.1	Field Documentation	150
5.19.1.1	ch_map	150
5.19.1.2	conn_hdl	150
5.19.1.3	status	150
5.20	LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM_T Struct Reference	150
5.20.1	Field Documentation	150
5.20.1.1	size	151
5.20.1.2	status	151
5.21	LE_CM_MSG_READ_RSSI_CFM_T Struct Reference	151
5.21.1	Field Documentation	151
5.21.1.1	conn_hdl	151
5.21.1.2	rssi	151
5.21.1.3	status	151
5.22	LE_CM_MSG_READ_TX_POWER_CFM_T Struct Reference	152
5.22.1	Field Documentation	152
5.22.1.1	conn_hdl	152
5.22.1.2	status	152
5.22.1.3	tx_power	152
5.23	LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM_T Struct Reference	152
5.23.1	Field Documentation	152

5.23.1.1	size	153
5.23.1.2	status	153
5.24	LE_CM_MSG_SET_DATA_LENGTH_CFM_T Struct Reference	153
5.24.1	Field Documentation	153
5.24.1.1	conn_hdl	153
5.24.1.2	status	153
5.25	LE_CM_MSG_SET_DISCONNECT_CFM_T Struct Reference	153
5.25.1	Field Documentation	154
5.25.1.1	handle	154
5.25.1.2	status	154
5.26	LE_CM_MSG_SIGNAL_UPDATE_REQ_T Struct Reference	154
5.26.1	Field Documentation	154
5.26.1.1	conn_hdl	154
5.26.1.2	identifier	155
5.26.1.3	interval_max	155
5.26.1.4	interval_min	155
5.26.1.5	slave_latency	155
5.26.1.6	timeout_multiplier	155
5.27	LE_CM_REQ_STATUS_T Struct Reference	155
5.27.1	Field Documentation	155
5.27.1.1	status	156
5.28	LE_CONN_PARA_T Struct Reference	156
5.28.1	Field Documentation	156
5.28.1.1	itv_max	156
5.28.1.2	itv_min	156
5.28.1.3	latency	156
5.28.1.4	sv_timeout	156
5.29	LE_GAP_ADVERTISING_PARAM_T Struct Reference	157
5.29.1	Field Documentation	157
5.29.1.1	channel_map	157

5.29.1.2	filter_policy	157
5.29.1.3	interval_max	157
5.29.1.4	interval_min	157
5.29.1.5	own_addr_type	158
5.29.1.6	peer_addr	158
5.29.1.7	peer_addr_type	158
5.29.1.8	type	158
5.30	LE_GAP_CONN_PARAM_T Struct Reference	158
5.30.1	Field Documentation	158
5.30.1.1	interval_max	158
5.30.1.2	interval_min	159
5.30.1.3	latency	159
5.30.1.4	supervision_timeout	159
5.31	LE_GAP_SCAN_PARAM_T Struct Reference	159
5.31.1	Field Documentation	159
5.31.1.1	filter_policy	159
5.31.1.2	interval	159
5.31.1.3	own_addr_type	160
5.31.1.4	type	160
5.31.1.5	window	160
5.32	LE_GATT_ATTR_T Struct Reference	160
5.32.1	Field Documentation	160
5.32.1.1	format	160
5.32.1.2	handle	161
5.32.1.3	len	161
5.32.1.4	maxLen	161
5.32.1.5	permit	161
5.32.1.6	pUuid	161
5.32.1.7	pVal	161
5.33	LE_GATT_MSG_ACCESS_READ_IND_T Struct Reference	161

5.33.1	Field Documentation	162
5.33.1.1	conn_hdl	162
5.33.1.2	devid	162
5.33.1.3	handle	162
5.33.1.4	offset	162
5.34	LE_GATT_MSG_ACCESS_WRITE_IND_T Struct Reference	162
5.34.1	Field Documentation	162
5.34.1.1	conn_hdl	163
5.34.1.2	devid	163
5.34.1.3	flag	163
5.34.1.4	handle	163
5.34.1.5	len	163
5.34.1.6	offset	163
5.34.1.7	pVal	163
5.35	LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T Struct Reference	163
5.35.1	Field Documentation	164
5.35.1.1	conn_hdl	164
5.35.1.2	devid	164
5.35.1.3	format	164
5.35.1.4	handle	164
5.35.1.5	uuid	164
5.36	LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T Struct Reference	164
5.36.1	Field Documentation	165
5.36.1.1	conn_hdl	165
5.36.1.2	devid	165
5.36.1.3	format	165
5.36.1.4	handle	165
5.36.1.5	property	165
5.36.1.6	uuid	166
5.36.1.7	val_hdl	166

5.37 LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T Struct Reference	166
5.37.1 Field Documentation	166
5.37.1.1 att_err	166
5.37.1.2 conn_hdl	166
5.37.1.3 devid	167
5.37.1.4 handle	167
5.37.1.5 len	167
5.37.1.6 offset	167
5.37.1.7 val	167
5.38 LE_GATT_MSG_CONFIRMATION_CFM_T Struct Reference	167
5.38.1 Field Documentation	167
5.38.1.1 conn_hdl	168
5.38.1.2 devid	168
5.38.1.3 handle	168
5.39 LE_GATT_MSG_EXCHANGE_MTU_CFM_T Struct Reference	168
5.39.1 Field Documentation	168
5.39.1.1 conn_hdl	168
5.39.1.2 current_rx_mtu	168
5.39.1.3 devid	169
5.40 LE_GATT_MSG_EXCHANGE_MTU_IND_T Struct Reference	169
5.40.1 Field Documentation	169
5.40.1.1 client_rx_mtu	169
5.40.1.2 conn_hdl	169
5.40.1.3 devid	169
5.41 LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T Struct Reference	169
5.41.1 Field Documentation	170
5.41.1.1 att_err	170
5.41.1.2 conn_hdl	170
5.41.1.3 devid	170
5.41.1.4 err_hdl	170

5.41.1.5	status	170
5.42	LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T Struct Reference	170
5.42.1	Field Documentation	171
5.42.1.1	att_err	171
5.42.1.2	conn_hdl	171
5.42.1.3	devid	171
5.42.1.4	handle	171
5.42.1.5	status	171
5.43	LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T Struct Reference	171
5.43.1	Field Documentation	172
5.43.1.1	att_err	172
5.43.1.2	conn_hdl	172
5.43.1.3	devid	172
5.43.1.4	handle	172
5.43.1.5	status	172
5.44	LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T Struct Reference	172
5.44.1	Field Documentation	173
5.44.1.1	att_err	173
5.44.1.2	conn_hdl	173
5.44.1.3	devid	173
5.44.1.4	handle	173
5.44.1.5	status	173
5.45	LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T Struct Reference	173
5.45.1	Field Documentation	174
5.45.1.1	att_err	174
5.45.1.2	conn_hdl	174
5.45.1.3	devid	174
5.45.1.4	handle	174
5.45.1.5	status	174
5.46	LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T Struct Reference	174

5.46.1	Field Documentation	175
5.46.1.1	att_err	175
5.46.1.2	conn_hdl	175
5.46.1.3	devid	175
5.46.1.4	handle	175
5.46.1.5	status	175
5.47	LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T Struct Reference	175
5.47.1	Field Documentation	176
5.47.1.1	conn_hdl	176
5.47.1.2	devid	176
5.47.1.3	end_hdl	176
5.47.1.4	format	176
5.47.1.5	handle	176
5.47.1.6	start_hdl	177
5.47.1.7	uuid	177
5.48	LE_GATT_MSG_INDICATE_IND_T Struct Reference	177
5.48.1	Field Documentation	177
5.48.1.1	conn_hdl	177
5.48.1.2	devid	177
5.48.1.3	handle	177
5.48.1.4	len	178
5.48.1.5	val	178
5.49	LE_GATT_MSG_NOTIFY_CFM_T Struct Reference	178
5.49.1	Field Documentation	178
5.49.1.1	conn_hdl	178
5.49.1.2	devid	178
5.49.1.3	handle	178
5.49.1.4	status	179
5.50	LE_GATT_MSG_NOTIFY_IND_T Struct Reference	179
5.50.1	Field Documentation	179

5.50.1.1	conn_hdl	179
5.50.1.2	devid	179
5.50.1.3	handle	179
5.50.1.4	len	179
5.50.1.5	val	180
5.51	LE_GATT_MSG_OPERATION_TIMEOUT_T Struct Reference	180
5.51.1	Field Documentation	180
5.51.1.1	att_op	180
5.51.1.2	conn_hdl	180
5.51.1.3	devid	180
5.52	LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T Struct Reference	180
5.52.1	Field Documentation	181
5.52.1.1	att_err	181
5.52.1.2	conn_hdl	181
5.52.1.3	devid	181
5.52.1.4	handle	181
5.52.1.5	status	181
5.53	LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T Struct Reference	181
5.53.1	Field Documentation	182
5.53.1.1	att_err	182
5.53.1.2	conn_hdl	182
5.53.1.3	devid	182
5.53.1.4	handle	182
5.53.1.5	status	182
5.54	LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T Struct Reference	182
5.54.1	Field Documentation	183
5.54.1.1	att_err	183
5.54.1.2	conn_hdl	183
5.54.1.3	devid	183
5.54.1.4	handle	183

5.54.1.5	status	183
5.55	LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T Struct Reference	183
5.55.1	Field Documentation	184
5.55.1.1	att_err	184
5.55.1.2	conn_hdl	184
5.55.1.3	devid	184
5.55.1.4	handle	184
5.55.1.5	status	184
5.56	LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T Struct Reference	184
5.56.1	Field Documentation	185
5.56.1.1	att_err	185
5.56.1.2	conn_hdl	185
5.56.1.3	devid	185
5.56.1.4	err_hdl	185
5.56.1.5	len	185
5.56.1.6	status	186
5.56.1.7	val	186
5.57	LE_GATT_MSG_SERVICE_INFO_IND_T Struct Reference	186
5.57.1	Field Documentation	186
5.57.1.1	conn_hdl	186
5.57.1.2	devid	186
5.57.1.3	end_hdl	186
5.57.1.4	format	187
5.57.1.5	start_hdl	187
5.57.1.6	uuid	187
5.58	LE_GATT_MSG_SIGNED_WRITE_CFM_T Struct Reference	187
5.58.1	Field Documentation	187
5.58.1.1	conn_hdl	187
5.58.1.2	devid	187
5.58.1.3	handle	188

5.58.1.4	status	188
5.59	LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T Struct Reference	188
5.59.1	Field Documentation	188
5.59.1.1	att_err	188
5.59.1.2	conn_hdl	188
5.59.1.3	devid	188
5.59.1.4	handle	189
5.59.1.5	status	189
5.60	LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T Struct Reference	189
5.60.1	Field Documentation	189
5.60.1.1	att_err	189
5.60.1.2	conn_hdl	189
5.60.1.3	devid	189
5.60.1.4	handle	190
5.60.1.5	status	190
5.61	LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T Struct Reference	190
5.61.1	Field Documentation	190
5.61.1.1	att_err	190
5.61.1.2	conn_hdl	190
5.61.1.3	devid	190
5.61.1.4	handle	191
5.61.1.5	status	191
5.62	LE_GATT_MSG_WRITE_NO_RSP_CFM_T Struct Reference	191
5.62.1	Field Documentation	191
5.62.1.1	conn_hdl	191
5.62.1.2	devid	191
5.62.1.3	handle	191
5.62.1.4	status	192
5.63	LE_GATT_SERVICE_T Struct Reference	192
5.63.1	Field Documentation	192

5.63.1.1	endHdl	192
5.63.1.2	pAttr	192
5.63.1.3	startHdl	192
5.63.1.4	svc_id	192
5.64	LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T Struct Reference	193
5.64.1	Field Documentation	193
5.64.1.1	conn_hdl	193
5.64.1.2	enable	193
5.65	LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T Struct Reference	193
5.65.1	Field Documentation	193
5.65.1.1	conn_hdl	193
5.65.1.2	status	194
5.66	LE_SMP_MSG_OOB_DATA_REQUEST_IND_T Struct Reference	194
5.66.1	Field Documentation	194
5.66.1.1	conn_hdl	194
5.67	LE_SMP_MSG_PAIRING_ACTION_IND_T Struct Reference	194
5.67.1	Field Documentation	194
5.67.1.1	action	194
5.67.1.2	conn_hdl	195
5.67.1.3	lost_bond	195
5.67.1.4	sc	195
5.68	LE_SMP_MSG_PAIRING_COMPLETE_IND_T Struct Reference	195
5.68.1	Field Documentation	195
5.68.1.1	authenticated	195
5.68.1.2	bonded	195
5.68.1.3	conn_hdl	196
5.68.1.4	peer_id_addr	196
5.68.1.5	sc	196
5.68.1.6	status	196
5.69	LE_SMP_MSG_PASSKEY_DISPLAY_IND_T Struct Reference	196

5.69.1	Field Documentation	196
5.69.1.1	conn_hdl	196
5.69.1.2	passkey	197
5.70	LE_SMP_MSG_PASSKEY_INPUT_IND_T Struct Reference	197
5.70.1	Field Documentation	197
5.70.1.1	conn_hdl	197
5.71	LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND_T Struct Reference	197
5.71.1	Field Documentation	197
5.71.1.1	conn_hdl	197
5.72	LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T Struct Reference	198
5.72.1	Field Documentation	198
5.72.1.1	bondable	198
5.72.1.2	conn_hdl	198
5.72.1.3	keypress	198
5.72.1.4	mitm	198
5.72.1.5	sc	198
5.73	LE_SMP_MSG_USER_CONFIRM_IND_T Struct Reference	199
5.73.1	Field Documentation	199
5.73.1.1	confirm_num	199
5.73.1.2	conn_hdl	199
5.74	LE_SMP_SC_OOB_DATA_T Struct Reference	199
5.74.1	Field Documentation	199
5.74.1.1	confirm	199
5.74.1.2	rand	200
5.75	LE_SYS_MSG_BUF_OVERFLOW_T Struct Reference	200
5.75.1	Field Documentation	200
5.75.1.1	conn_hdl	200
5.76	mw_wifi_auto_connect_ap_info_t Struct Reference	200
5.76.1	Field Documentation	201
5.76.1.1	ap_channel	201

5.76.1.2	beacon_interval	201
5.76.1.3	bssid	201
5.76.1.4	capabilities	201
5.76.1.5	dtim_prod	201
5.76.1.6	fast_connect	201
5.76.1.7	free_ocpy	201
5.76.1.8	hid_ssid	202
5.76.1.9	latest_beacon_rx_time	202
5.76.1.10	passphrase	202
5.76.1.11	psk	202
5.76.1.12	rsn_ie	202
5.76.1.13	rsi	202
5.76.1.14	ssid	202
5.76.1.15	supported_rates	202
5.76.1.16	wpa_data	203
5.76.1.17	wpa_ie	203
5.77	MwFimAutoConnectCFG_t Struct Reference	203
5.77.1	Field Documentation	203
5.77.1.1	flag	203
5.77.1.2	front	203
5.77.1.3	max_save_num	203
5.77.1.4	rear	204
5.77.1.5	targetIdx	204
5.78	T_RfCmd Struct Reference	204
5.78.1	Field Documentation	204
5.78.1.1	iArgc	204
5.78.1.2	saArgv	204
5.78.1.3	u32Type	204
5.79	T_RfEvt Struct Reference	204
5.79.1	Field Documentation	205

5.79.1.1	pParam	205
5.79.1.2	u16RfMode	205
5.79.1.3	u16RxCnt	205
5.79.1.4	u16RxCrcOkCnt	205
5.79.1.5	u32Freq	206
5.79.1.6	u32Mode	206
5.79.1.7	u32RfChannel	206
5.79.1.8	u32Type	206
5.79.1.9	u8Freq	206
5.79.1.10	u8IpcEnable	206
5.79.1.11	u8Len	206
5.79.1.12	u8Pkt	206
5.79.1.13	u8Reserved	207
5.79.1.14	u8Status	207
5.79.1.15	u8Unicast	207
5.80	wifi_active_scan_time_t Struct Reference	207
5.80.1	Detailed Description	207
5.80.2	Field Documentation	207
5.80.2.1	max	207
5.80.2.2	min	208
5.81	wifi_ap_config_t Struct Reference	208
5.81.1	Detailed Description	208
5.81.2	Field Documentation	208
5.81.2.1	auth_mode	208
5.81.2.2	beacon_interval	208
5.81.2.3	channel	209
5.81.2.4	encrypt_type	209
5.81.2.5	max_connection	209
5.81.2.6	password	209
5.81.2.7	password_length	209

5.81.2.8	ssid	209
5.81.2.9	ssid_hidden	209
5.81.2.10	ssid_length	209
5.82	wifi_auto_connect_info_t Struct Reference	210
5.82.1	Detailed Description	210
5.82.2	Field Documentation	210
5.82.2.1	ap_channel	210
5.82.2.2	beacon_interval	210
5.82.2.3	bssid	211
5.82.2.4	capabilities	211
5.82.2.5	dtim_prod	211
5.82.2.6	fast_connect	211
5.82.2.7	free_ocpy	211
5.82.2.8	hid_ssid	211
5.82.2.9	latest_beacon_rx_time	211
5.82.2.10	passphrase	211
5.82.2.11	psk	212
5.82.2.12	rsn_ie	212
5.82.2.13	rsi	212
5.82.2.14	ssid	212
5.82.2.15	supported_rates	212
5.82.2.16	wpa_data	212
5.82.2.17	wpa_ie	212
5.83	wifi_config_t Union Reference	212
5.83.1	Detailed Description	213
5.83.2	Field Documentation	213
5.83.2.1	ap_config	213
5.83.2.2	sta_config	213
5.84	wifi_event_info_t Union Reference	213
5.84.1	Detailed Description	213

5.84.2	Field Documentation	214
5.84.2.1	connected	214
5.84.2.2	disconnected	214
5.84.2.3	got_ip	214
5.84.2.4	scan_done	214
5.85	wifi_event_sta_connected_t Struct Reference	214
5.85.1	Detailed Description	214
5.85.2	Field Documentation	215
5.85.2.1	authmode	215
5.85.2.2	bssid	215
5.85.2.3	channel	215
5.85.2.4	ssid	215
5.85.2.5	ssid_len	215
5.86	wifi_event_sta_disconnected_t Struct Reference	215
5.86.1	Detailed Description	216
5.86.2	Field Documentation	216
5.86.2.1	bssid	216
5.86.2.2	reason	216
5.86.2.3	ssid	216
5.86.2.4	ssid_len	216
5.87	wifi_event_sta_got_ip_t Struct Reference	216
5.87.1	Detailed Description	217
5.87.2	Field Documentation	217
5.87.2.1	ip_changed	217
5.88	wifi_event_sta_scan_done_t Struct Reference	217
5.88.1	Detailed Description	217
5.88.2	Field Documentation	217
5.88.2.1	number	217
5.88.2.2	scan_id	217
5.88.2.3	status	218

5.89	wifi_fast_scan_threshold_t Struct Reference	218
5.89.1	Detailed Description	218
5.89.2	Field Documentation	218
5.89.2.1	authmode	218
5.89.2.2	rsi	218
5.90	wifi_init_config_t Struct Reference	218
5.90.1	Detailed Description	219
5.90.2	Field Documentation	219
5.90.2.1	event_handler	219
5.90.2.2	magic	219
5.91	wifi_scan_config_t Struct Reference	219
5.91.1	Detailed Description	219
5.91.2	Field Documentation	220
5.91.2.1	bssid	220
5.91.2.2	channel	220
5.91.2.3	scan_time	220
5.91.2.4	scan_type	220
5.91.2.5	show_hidden	220
5.91.2.6	ssid	220
5.92	wifi_scan_info_t Struct Reference	220
5.92.1	Detailed Description	221
5.92.2	Field Documentation	221
5.92.2.1	auth_mode	221
5.92.2.2	beacon_interval	221
5.92.2.3	bssid	221
5.92.2.4	capability_info	221
5.92.2.5	channel	222
5.92.2.6	group_cipher	222
5.92.2.7	pairwise_cipher	222
5.92.2.8	rsi	222

5.92.2.9	ssid	222
5.92.2.10	ssid_length	222
5.93	wifi_scan_list_t Struct Reference	222
5.93.1	Detailed Description	223
5.93.2	Field Documentation	223
5.93.2.1	ap_record	223
5.93.2.2	num	223
5.94	wifi_scan_time_t Union Reference	223
5.94.1	Detailed Description	223
5.94.2	Field Documentation	223
5.94.2.1	active	223
5.94.2.2	passive	224
5.95	wifi_sta_config_t Struct Reference	224
5.95.1	Detailed Description	224
5.95.2	Field Documentation	224
5.95.2.1	bssid	224
5.95.2.2	bssid_present	224
5.95.2.3	password	225
5.95.2.4	password_length	225
5.95.2.5	scan_method	225
5.95.2.6	sort_method	225
5.95.2.7	ssid	225
5.95.2.8	ssid_length	225
5.95.2.9	threshold	225
Index		227

Chapter 1

SDK PREVIEW

- BLE APIs :
 - GAP APIs : BLE GAP APIs
 - GATT APIs : BLE GATT APIs
 - CM APIs : BLE CM APIs
 - MSG APIs : BLE MSG APIs
 - SMP APIs : BLE SMP APIs
- WiFi APIs :
 - Station APIs : STATION APIs
 - Common APIs : COMMON APIs
 - Enumerations : ENUMERATIONS

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

BLE ALL APIs	9
BLE CM APIs	10
BLE GAP APIs	16
BLE GATT APIs	37
BLE MSG APIs	71
BLE SMP APIs	83
WIFI APIs	91
WIFI Common APIs	96
WIFI STA APIs	100
Enumeration	125

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

auto_conn_info_t	131
auto_connect_cfg_t	134
event_msg_t	
Send information to event by event_msg_t	135
hap_control_t	136
LE_BT_ADDR_T	137
LE_CM_CONNECTION_COMPLETE_IND_T	138
LE_CM_MSG_ADVERTISE_REPORT_IND_T	139
LE_CM_MSG_CONN_PARA_REQ_T	140
LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T	141
LE_CM_MSG_DATA_LEN_CHANGE_IND_T	142
LE_CM_MSG_DIRECT_ADV_REPORT_IND_T	143
LE_CM_MSG_DISCONNECT_COMPLETE_IND_T	144
LE_CM_MSG_ENCRYPTION_CHANGE_IND_T	145
LE_CM_MSG_ENCRYPTION_REFRESH_IND_T	146
LE_CM_MSG_INIT_COMPLETE_CFM_T	147
LE_CM_MSG_LTK_REQ_IND_T	147
LE_CM_MSG_READ_ADV_TX_POWER_CFM_T	148
LE_CM_MSG_READ_BD_ADDR_CFM_T	149
LE_CM_MSG_READ_CHANNEL_MAP_CFM_T	150
LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM_T	150
LE_CM_MSG_READ_RSSI_CFM_T	151
LE_CM_MSG_READ_TX_POWER_CFM_T	152
LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM_T	152
LE_CM_MSG_SET_DATA_LENGTH_CFM_T	153
LE_CM_MSG_SET_DISCONNECT_CFM_T	153
LE_CM_MSG_SIGNAL_UPDATE_REQ_T	154
LE_CM_REQ_STATUS_T	155
LE_CONN_PARA_T	156
LE_GAP_ADVERTISING_PARAM_T	157
LE_GAP_CONN_PARAM_T	158
LE_GAP_SCAN_PARAM_T	159
LE_GATT_ATTR_T	160
LE_GATT_MSG_ACCESS_READ_IND_T	161
LE_GATT_MSG_ACCESS_WRITE_IND_T	162

LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T	163
LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T	164
LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T	166
LE_GATT_MSG_CONFIRMATION_CFM_T	167
LE_GATT_MSG_EXCHANGE_MTU_CFM_T	168
LE_GATT_MSG_EXCHANGE_MTU_IND_T	169
LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T	169
LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T	170
LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T	171
LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T	172
LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T	173
LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T	174
LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T	175
LE_GATT_MSG_INDICATE_IND_T	177
LE_GATT_MSG_NOTIFY_CFM_T	178
LE_GATT_MSG_NOTIFY_IND_T	179
LE_GATT_MSG_OPERATION_TIMEOUT_T	180
LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T	180
LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T	181
LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T	182
LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T	183
LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T	184
LE_GATT_MSG_SERVICE_INFO_IND_T	186
LE_GATT_MSG_SIGNED_WRITE_CFM_T	187
LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T	188
LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T	189
LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T	190
LE_GATT_MSG_WRITE_NO_RSP_CFM_T	191
LE_GATT_SERVICE_T	192
LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T	193
LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T	193
LE_SMP_MSG_OOB_DATA_REQUEST_IND_T	194
LE_SMP_MSG_PAIRING_ACTION_IND_T	194
LE_SMP_MSG_PAIRING_COMPLETE_IND_T	195
LE_SMP_MSG_PASSKEY_DISPLAY_IND_T	196
LE_SMP_MSG_PASSKEY_INPUT_IND_T	197
LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND_T	197
LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T	198
LE_SMP_MSG_USER_CONFIRM_IND_T	199
LE_SMP_SC_OOB_DATA_T	199
LE_SYS_MSG_BUF_OVERFLOW_T	200
mw_wifi_auto_connect_ap_info_t	200
MwFimAutoConnectCFG_t	203
T_RfCmd	204
T_RfEvt	204
wifi_active_scan_time_t	
Range of active scan times per channel	207
wifi_ap_config_t	
This structure is the Wi-Fi configuration for initialization for Soft-AP mode	208
wifi_auto_connect_info_t	
WiFi auto connect info parameters	210
wifi_config_t	
Wi-Fi configuration for initialization	212
wifi_event_info_t	
Wifi_event_info_t	213
wifi_event_sta_connected_t	
Wifi_event_sta_connected_t	214

wifi_event_sta_disconnected_t	
Wifi_event_sta_disconnected_t	215
wifi_event_sta_got_ip_t	
Wifi_event_sta_got_ip_t	216
wifi_event_sta_scan_done_t	
Wifi_event_sta_scan_done_t	217
wifi_fast_scan_threshold_t	
Structure describing parameters for a Wi-Fi fast scan	218
wifi_init_config_t	
WiFi stack configuration parameters	218
wifi_scan_config_t	
Parameters for an SSID scan	219
wifi_scan_info_t	
This structure defines the information of scanned APs	220
wifi_scan_list_t	
This structure defines the list of scanned APs with their corresponding information	222
wifi_scan_time_t	
Aggregate of active & passive scan time per channel	223
wifi_sta_config_t	
This structure is the Wi-Fi configuration for initialization for STA mode	224

Chapter 4

Module Documentation

4.1 BLE ALL APIs

BLE ALL APIs.

Modules

- [BLE CM APIs](#)
- [BLE GAP APIs](#)
- [BLE GATT APIs](#)
- [BLE MSG APIs](#)
- [BLE SMP APIs](#)

4.1.1 Detailed Description

BLE ALL APIs.

4.2 BLE CM APIs

Data Structures

- struct [LE_CM_CONNECTION_COMPLETE_IND_T](#)
- struct [LE_CM_MSG_ADVERTISE_REPORT_IND_T](#)
- struct [LE_CM_MSG_CONN_PARA_REQ_T](#)
- struct [LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T](#)
- struct [LE_CM_MSG_DATA_LEN_CHANGE_IND_T](#)
- struct [LE_CM_MSG_DIRECT_ADV_REPORT_IND_T](#)
- struct [LE_CM_MSG_DISCONNECT_COMPLETE_IND_T](#)
- struct [LE_CM_MSG_ENCRYPTION_CHANGE_IND_T](#)
- struct [LE_CM_MSG_ENCRYPTION_REFRESH_IND_T](#)
- struct [LE_CM_MSG_INIT_COMPLETE_CFM_T](#)
- struct [LE_CM_MSG_LTK_REQ_IND_T](#)
- struct [LE_CM_MSG_READ_ADV_TX_POWER_CFM_T](#)
- struct [LE_CM_MSG_READ_BD_ADDR_CFM_T](#)
- struct [LE_CM_MSG_READ_CHANNEL_MAP_CFM_T](#)
- struct [LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM_T](#)
- struct [LE_CM_MSG_READ_RSSI_CFM_T](#)
- struct [LE_CM_MSG_READ_TX_POWER_CFM_T](#)
- struct [LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM_T](#)
- struct [LE_CM_MSG_SET_DATA_LENGTH_CFM_T](#)
- struct [LE_CM_MSG_SET_DISCONNECT_CFM_T](#)
- struct [LE_CM_MSG_SIGNAL_UPDATE_REQ_T](#)
- struct [LE_CM_REQ_STATUS_T](#)

Typedefs

- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_ADD_TO_WHITE_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_CANCEL_CONNECTION_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_CREATE_CONNECTION_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_ENTER_ADVERTISING_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_ENTER_SCANNING_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_EXIT_ADVERTISING_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_EXIT_SCANNING_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_CHANNEL_MAP_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_SCAN_PARAMS_CFM_T](#)
- typedef [LE_CM_REQ_STATUS_T](#) [LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T](#)

Enumerations

- enum {
[LE_CM_MSG_INIT_COMPLETE_CFM](#) = [LE_CM_MSG_BASE](#), [LE_CM_MSG_SET_DISCONNECT_CFM](#),
[LE_CM_MSG_DISCONNECT_COMPLETE_IND](#), [LE_CM_MSG_SET_ADVERTISING_DATA_CFM](#),
[LE_CM_MSG_SET_SCAN_RSP_DATA_CFM](#), [LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM](#),
[LE_CM_MSG_ENTER_ADVERTISING_CFM](#), [LE_CM_MSG_EXIT_ADVERTISING_CFM](#),
[LE_CM_MSG_SET_SCAN_PARAMS_CFM](#), [LE_CM_MSG_ENTER_SCANNING_CFM](#), [LE_CM_MSG_EXIT_SCANNING_CFM](#),
[LE_CM_MSG_CREATE_CONNECTION_CFM](#),
[LE_CM_MSG_CANCEL_CONNECTION_CFM](#), [LE_CM_MSG_READ_TX_POWER_CFM](#), [LE_CM_MSG_READ_BD_ADDR_CFM](#),
[LE_CM_MSG_READ_RSSI_CFM](#),
[LE_CM_MSG_SET_RANDOM_ADDRESS_CFM](#), [LE_CM_MSG_READ_ADV_TX_POWER_CFM](#), [LE_CM_MSG_READ_WHITE_LIST_CFM](#),
[LE_CM_MSG_CLEAR_WHITE_LIST_CFM](#),
[LE_CM_MSG_ADD_TO_WHITE_LIST_CFM](#), [LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM](#),
[LE_CM_MSG_SET_CHANNEL_MAP_CFM](#), [LE_CM_MSG_READ_CHANNEL_MAP_CFM](#),
[LE_CM_MSG_SET_DATA_LENGTH_CFM](#), [LE_CM_MSG_DATA_LEN_CHANGE_IND](#), [LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM](#),
[LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM](#),
[LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM](#), [LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM](#),
[LE_CM_MSG_SET_RPA_TIMEOUT_CFM](#), [LE_CM_MSG_SIGNAL_UPDATE_REQ](#),
[LE_CM_MSG_CONN_UPDATE_COMPLETE_IND](#), [LE_CM_MSG_CONN_PARAM_REQ](#), [LE_CM_MSG_ENCRYPTION_CHANNEL_REQ](#),
[LE_CM_MSG_ENCRYPTION_REFRESH_IND](#),
[LE_CM_MSG_LTK_REQ_IND](#), [LE_CM_MSG_ADVERTISE_REPORT_IND](#), [LE_CM_MSG_DIRECT_ADV_REPORT_IND](#),
[LE_CM_CONNECTION_COMPLETE_IND](#),
[LE_CM_MSG_READ_LOCAL_RPA_CFM](#), [LE_CM_MSG_TOP](#) }

BLE connection management message id.

Functions

- void [LeCmInit](#) ([TASK](#) appTask)
BLE Connection Management Module Init.

4.2.1 Detailed Description

4.2.2 Typedef Documentation

4.2.2.1 [LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM_T](#)

```
typedef LE\_CM\_REQ\_STATUS\_T LE\_CM\_MSG\_ADD\_TO\_RESOLVING\_LIST\_CFM\_T
```

4.2.2.2 [LE_CM_MSG_ADD_TO_WHITE_LIST_CFM_T](#)

```
typedef LE\_CM\_REQ\_STATUS\_T LE\_CM\_MSG\_ADD\_TO\_WHITE\_LIST\_CFM\_T
```

4.2.2.3 LE_CM_MSG_CANCEL_CONNECTION_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_CANCEL_CONNECTION_CFM_T
```

4.2.2.4 LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM_T
```

4.2.2.5 LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T
```

4.2.2.6 LE_CM_MSG_CREATE_CONNECTION_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_CREATE_CONNECTION_CFM_T
```

4.2.2.7 LE_CM_MSG_ENTER_ADVERTISING_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_ENTER_ADVERTISING_CFM_T
```

4.2.2.8 LE_CM_MSG_ENTER_SCANNING_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_ENTER_SCANNING_CFM_T
```

4.2.2.9 LE_CM_MSG_EXIT_ADVERTISING_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_EXIT_ADVERTISING_CFM_T
```

4.2.2.10 LE_CM_MSG_EXIT_SCANNING_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_EXIT_SCANNING_CFM_T
```

4.2.2.11 LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T
```

4.2.2.12 LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T
```

4.2.2.13 LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T
```

4.2.2.14 LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T
```

4.2.2.15 LE_CM_MSG_SET_CHANNEL_MAP_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_CHANNEL_MAP_CFM_T
```

4.2.2.16 LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T
```

4.2.2.17 LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T
```

4.2.2.18 LE_CM_MSG_SET_SCAN_PARAMS_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_SCAN_PARAMS_CFM_T
```

4.2.2.19 LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T

```
typedef LE_CM_REQ_STATUS_T LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T
```

4.2.3 Enumeration Type Documentation

4.2.3.1 anonymous enum

```
anonymous enum
```

BLE connection management message id.

Enumerator

LE_CM_MSG_INIT_COMPLETE_CFM	initialize complete
LE_CM_MSG_SET_DISCONNECT_CFM	set disconnect confirm
LE_CM_MSG_DISCONNECT_COMPLETE_IND	disconnect complete indication
LE_CM_MSG_SET_ADVERTISING_DATA_CFM	set advertising data confirm
LE_CM_MSG_SET_SCAN_RSP_DATA_CFM	set scan response data confirm
LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM	set advertising parameters confirm
LE_CM_MSG_ENTER_ADVERTISING_CFM	enter advertising confirm
LE_CM_MSG_EXIT_ADVERTISING_CFM	exit advertising confirm
LE_CM_MSG_SET_SCAN_PARAMS_CFM	set scan parameters confirm
LE_CM_MSG_ENTER_SCANNING_CFM	enter scanning confirm
LE_CM_MSG_EXIT_SCANNING_CFM	exit scanning confirm
LE_CM_MSG_CREATE_CONNECTION_CFM	create connection confirm
LE_CM_MSG_CANCEL_CONNECTION_CFM	cancel connection confirm
LE_CM_MSG_READ_TX_POWER_CFM	read tx power confirm
LE_CM_MSG_READ_BD_ADDR_CFM	read device address confirm
LE_CM_MSG_READ_RSSI_CFM	read RSSI confirm
LE_CM_MSG_SET_RANDOM_ADDRESS_CFM	set random address confirm
LE_CM_MSG_READ_ADV_TX_POWER_CFM	read advertising tx power confirm
LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM	read whitelist size confirm
LE_CM_MSG_CLEAR_WHITE_LIST_CFM	clear whitelist confirm
LE_CM_MSG_ADD_TO_WHITE_LIST_CFM	add to whitelist confirm
LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM	remove from whitelist confirm
LE_CM_MSG_SET_CHANNEL_MAP_CFM	set channel map confirm
LE_CM_MSG_READ_CHANNEL_MAP_CFM	read channel map confirm
LE_CM_MSG_SET_DATA_LENGTH_CFM	set data length confirm
LE_CM_MSG_DATA_LEN_CHANGE_IND	data length change indication
LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM	add to resolving list confirm
LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM	remove from resolving list confirm
LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM	clear resolving list confirm
LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM	read resolving list size confirm
LE_CM_MSG_SET_RPA_TIMEOUT_CFM	set resolving private address timeout confirm
LE_CM_MSG_SIGNAL_UPDATE_REQ	signal update request

Enumerator

LE_CM_MSG_CONN_UPDATE_COMPLETE_IND	connection update complete indication
LE_CM_MSG_CONN_PARA_REQ	connection parameters request
LE_CM_MSG_ENCRYPTION_CHANGE_IND	encryption change indication
LE_CM_MSG_ENCRYPTION_REFRESH_IND	encryption refresh indication
LE_CM_MSG_LTK_REQ_IND	long term key indication
LE_CM_MSG_ADVERTISE_REPORT_IND	advertising report indication
LE_CM_MSG_DIRECT_ADV_REPORT_IND	direct advertising report indication
LE_CM_CONNECTION_COMPLETE_IND	connection complete indication
LE_CM_MSG_READ_LOCAL_RPA_CFM	read local resolving private address confirm
LE_CM_MSG_TOP	top of CM message id

4.2.4 Function Documentation

4.2.4.1 LeCmInit()

```
void LeCmInit (
    TASK appTask )
```

BLE Connection Management Module Init.

Parameters

<i>the</i>	reference of BLE task.
------------	------------------------

Returns

None.

4.3 BLE GAP APIs

Data Structures

- struct [LE_GAP_ADVERTISING_PARAM_T](#)
- struct [LE_GAP_CONN_PARAM_T](#)
- struct [LE_GAP_SCAN_PARAM_T](#)

Macros

- #define [GAP_ADTYPE_128BIT_COMPLETE](#) 0x07
- #define [GAP_ADTYPE_128BIT_MORE](#) 0x06
- #define [GAP_ADTYPE_16BIT_COMPLETE](#) 0x03
- #define [GAP_ADTYPE_16BIT_MORE](#) 0x02
- #define [GAP_ADTYPE_32BIT_COMPLETE](#) 0x05
- #define [GAP_ADTYPE_32BIT_MORE](#) 0x04
- #define [GAP_ADTYPE_3D_INFO_DATA](#) 0x3D
- #define [GAP_ADTYPE_ADV_INTERVAL](#) 0x1A
- #define [GAP_ADTYPE_APPEARANCE](#) 0x19
- #define [GAP_ADTYPE_FLAGS](#) 0x01
- #define [GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED](#) 0x04
- #define [GAP_ADTYPE_FLAGS_GENERAL](#) 0x02
- #define [GAP_ADTYPE_FLAGS_LIMITED](#) 0x01
- #define [GAP_ADTYPE_LE_BD_ADDR](#) 0x1B
- #define [GAP_ADTYPE_LE_ROLE](#) 0x1C
- #define [GAP_ADTYPE_LOCAL_NAME_COMPLETE](#) 0x09
- #define [GAP_ADTYPE_LOCAL_NAME_SHORT](#) 0x08
- #define [GAP_ADTYPE_MANUFACTURER_SPECIFIC](#) 0xFF
- #define [GAP_ADTYPE_OOB_CLASS_OF_DEVICE](#) 0x0D
- #define [GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC](#) 0x0E
- #define [GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDOM](#) 0x0F
- #define [GAP_ADTYPE_POWER_LEVEL](#) 0x0A
- #define [GAP_ADTYPE_PUBLIC_TARGET_ADDR](#) 0x17
- #define [GAP_ADTYPE_RANDOM_TARGET_ADDR](#) 0x18
- #define [GAP_ADTYPE_SERVICE_DATA](#) 0x16
- #define [GAP_ADTYPE_SERVICE_DATA_128BIT](#) 0x21
- #define [GAP_ADTYPE_SERVICE_DATA_32BIT](#) 0x20
- #define [GAP_ADTYPE_SERVICES_LIST_128BIT](#) 0x15
- #define [GAP_ADTYPE_SERVICES_LIST_16BIT](#) 0x14
- #define [GAP_ADTYPE_SIGNED_DATA](#) 0x13
- #define [GAP_ADTYPE_SIMPLE_PAIRING_HASHC_256](#) 0x1D
- #define [GAP_ADTYPE_SIMPLE_PAIRING_RANDOM_256](#) 0x1E
- #define [GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE](#) 0x12
- #define [GAP_ADTYPE_SM_OOB_FLAG](#) 0x11
- #define [GAP_ADTYPE_SM_TK](#) 0x10
- #define [GAP_PUBLIC_ADDR](#) 0
- #define [GAP_RAND_ADDR_NRPA](#) 2
- #define [GAP_RAND_ADDR_RPA](#) 3
- #define [GAP_RAND_ADDR_STATIC](#) 1
- #define [GAP_SCAN_TYPE_ACTIVE](#) 1
- #define [GAP_SCAN_TYPE_PASSIVE](#) 0
- #define [GAP_TX_PWR_CURR_VAL](#) 0
- #define [GAP_TX_PWR_MAX_VAL](#) 1

- #define `GAPBOND_IO_CAP_DISPLAY_ONLY` 0x00
- #define `GAPBOND_IO_CAP_DISPLAY_YES_NO` 0x01
- #define `GAPBOND_IO_CAP_KEYBOARD_DISPLAY` 0x04
- #define `GAPBOND_IO_CAP_KEYBOARD_ONLY` 0x02
- #define `GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT` 0x03
- #define `GAPBOND_PAIRING_MODE_INITIATE` 0x02
- #define `GAPBOND_PAIRING_MODE_NO_PAIRING` 0x00
- #define `GAPBOND_PAIRING_MODE_WAIT_FOR_REQ` 0x01
- #define `LE_GAP_ADV_MAX_SIZE` 31

Functions

- `LE_ERR_STATE` `LeGapAddToResolvingList` (`LE_BT_ADDR_T` *bt_addr, `UINT8` *irk)
Add device to resolving-list.
- `LE_ERR_STATE` `LeGapAddToWhiteList` (`LE_BT_ADDR_T` *bt_addr)
Add device to whitelist.
- `LE_ERR_STATE` `LeGapAdvertisingEnable` (`BOOL` start)
Enable or disable advertising function.
- `LE_ERR_STATE` `LeGapCentralConnectReq` (`LE_BT_ADDR_T` *taddr, `UINT8` own_addr_type)
Central connect request.
- `LE_ERR_STATE` `LeGapCentralSetDataChannel` (`UINT8` *ch)
Central set data channel.
- `LE_ERR_STATE` `LeGapClearResolvingList` (`void`)
Clear the resolving-list in the controller.
- `LE_ERR_STATE` `LeGapClearWhiteList` (`void`)
Clear whitelist in the controller.
- `LE_ERR_STATE` `LeGapConnectCancelReq` (`void`)
Cancel connect request.
- `void` `LeGapConnParaRequestRsp` (`UINT16` conn_hdl, `BOOL` accept)
Connection parameters request response.
- `void` `LeGapConnUpdateRequest` (`UINT16` conn_hdl, `LE_CONN_PARA_T` *para)
Connection parameters update request.
- `void` `LeGapConnUpdateResponse` (`UINT16` conn_hdl, `UINT8` identifier, `BOOL` accept)
Connection parameters update response.
- `LE_ERR_STATE` `LeGapDisconnectReq` (`UINT16` conn_hdl)
Disconnect the physical connection.
- `LE_ERR_STATE` `LeGapGenRandAddr` (`UINT8` type, `BD_ADDR` addr)
Called to generation random address.
- `void` `LeGapGetBtAddr` (`void`)
Get owner device address.
- `void` `LeGapReadAdvChannelTxPower` (`void`)
Read ADV channel txpower.
- `LE_ERR_STATE` `LeGapReadChannelMap` (`UINT16` conn_hdl)
Read channel map.
- `void` `LeGapReadResolvingListSize` (`void`)
Read the resolving-list size in the controller.
- `LE_ERR_STATE` `LeGapReadRssi` (`UINT16` conn_hdl)
Read RSSI value from controller.
- `LE_ERR_STATE` `LeGapReadTxPower` (`UINT16` conn_hdl, `UINT8` type)
Read tx power value for the specified connection.
- `void` `LeGapReadWhiteListSize` (`void`)

- Read whitelist size in the controller.*

 - LE_ERR_STATE [LeGapRemoveFromWhiteList](#) (LE_BT_ADDR_T *bt_addr)

Remove device from whitelist.
- LE_ERR_STATE [LeGapScanningReq](#) (BOOL start, BOOL filter)

Request scanning start.
- LE_ERR_STATE [LeGapSetAdvData](#) (UINT8 len, UINT8 *data)

Called to set ADV data.
- LE_ERR_STATE [LeGapSetAdvParameter](#) (LE_GAP_ADVERTISING_PARAM_T *params)

Called to set ADV parameters.
- LE_ERR_STATE [LeGapSetConnParameter](#) (UINT16 interval_min, UINT16 interval_max, UINT16 slave_latency, UINT16 supervision_timeout)

Called to set connection parameters.
- LE_ERR_STATE [LeGapSetDataChannelPduLen](#) (UINT16 conn_hdl, UINT16 tx_octets, UINT16 tx_time)

Set data channel PDU length.
- LE_ERR_STATE [LeGapSetRandAddr](#) (BD_ADDR addr)

Called to set random address.
- LE_ERR_STATE [LeGapSetRpaTimeout](#) (UINT16 timeout)

Set resolvable private address timeout.
- LE_ERR_STATE [LeGapSetStaticAddr](#) (BD_ADDR addr)

Called to set static address.
- LE_ERR_STATE [LeSetScanParameter](#) (LE_GAP_SCAN_PARAM_T *params)

Called to set scan parameters.
- LE_ERR_STATE [LeSetScanRspData](#) (UINT8 len, UINT8 *data)

Called to set scan response data.

4.3.1 Detailed Description

4.3.2 Macro Definition Documentation

4.3.2.1 GAP_ADTYPE_128BIT_COMPLETE

```
#define GAP_ADTYPE_128BIT_COMPLETE 0x07
```

4.3.2.2 GAP_ADTYPE_128BIT_MORE

```
#define GAP_ADTYPE_128BIT_MORE 0x06
```

4.3.2.3 GAP_ADTYPE_16BIT_COMPLETE

```
#define GAP_ADTYPE_16BIT_COMPLETE 0x03
```


4.3.2.4 GAP_ADTYPE_16BIT_MORE

```
#define GAP_ADTYPE_16BIT_MORE 0x02
```

4.3.2.5 GAP_ADTYPE_32BIT_COMPLETE

```
#define GAP_ADTYPE_32BIT_COMPLETE 0x05
```

4.3.2.6 GAP_ADTYPE_32BIT_MORE

```
#define GAP_ADTYPE_32BIT_MORE 0x04
```

4.3.2.7 GAP_ADTYPE_3D_INFO_DATA

```
#define GAP_ADTYPE_3D_INFO_DATA 0x3D
```

4.3.2.8 GAP_ADTYPE_ADV_INTERVAL

```
#define GAP_ADTYPE_ADV_INTERVAL 0x1A
```

4.3.2.9 GAP_ADTYPE_APPEARANCE

```
#define GAP_ADTYPE_APPEARANCE 0x19
```

4.3.2.10 GAP_ADTYPE_FLAGS

```
#define GAP_ADTYPE_FLAGS 0x01
```

4.3.2.11 GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED

```
#define GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED 0x04
```

4.3.2.12 GAP_ADTYPE_FLAGS_GENERAL

```
#define GAP_ADTYPE_FLAGS_GENERAL 0x02
```

4.3.2.13 GAP_ADTYPE_FLAGS_LIMITED

```
#define GAP_ADTYPE_FLAGS_LIMITED 0x01
```

4.3.2.14 GAP_ADTYPE_LE_BD_ADDR

```
#define GAP_ADTYPE_LE_BD_ADDR 0x1B
```

4.3.2.15 GAP_ADTYPE_LE_ROLE

```
#define GAP_ADTYPE_LE_ROLE 0x1C
```

4.3.2.16 GAP_ADTYPE_LOCAL_NAME_COMPLETE

```
#define GAP_ADTYPE_LOCAL_NAME_COMPLETE 0x09
```

4.3.2.17 GAP_ADTYPE_LOCAL_NAME_SHORT

```
#define GAP_ADTYPE_LOCAL_NAME_SHORT 0x08
```

4.3.2.18 GAP_ADTYPE_MANUFACTURER_SPECIFIC

```
#define GAP_ADTYPE_MANUFACTURER_SPECIFIC 0xFF
```

4.3.2.19 GAP_ADTYPE_OOB_CLASS_OF_DEVICE

```
#define GAP_ADTYPE_OOB_CLASS_OF_DEVICE 0x0D
```

4.3.2.20 GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC

```
#define GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC 0x0E
```

4.3.2.21 GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDR

```
#define GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDR 0x0F
```

4.3.2.22 GAP_ADTYPE_POWER_LEVEL

```
#define GAP_ADTYPE_POWER_LEVEL 0x0A
```

4.3.2.23 GAP_ADTYPE_PUBLIC_TARGET_ADDR

```
#define GAP_ADTYPE_PUBLIC_TARGET_ADDR 0x17
```

4.3.2.24 GAP_ADTYPE_RANDOM_TARGET_ADDR

```
#define GAP_ADTYPE_RANDOM_TARGET_ADDR 0x18
```

4.3.2.25 GAP_ADTYPE_SERVICE_DATA

```
#define GAP_ADTYPE_SERVICE_DATA 0x16
```

4.3.2.26 GAP_ADTYPE_SERVICE_DATA_128BIT

```
#define GAP_ADTYPE_SERVICE_DATA_128BIT 0x21
```

4.3.2.27 GAP_ADTYPE_SERVICE_DATA_32BIT

```
#define GAP_ADTYPE_SERVICE_DATA_32BIT 0x20
```

4.3.2.28 GAP_ADTYPE_SERVICES_LIST_128BIT

```
#define GAP_ADTYPE_SERVICES_LIST_128BIT 0x15
```

4.3.2.29 GAP_ADTYPE_SERVICES_LIST_16BIT

```
#define GAP_ADTYPE_SERVICES_LIST_16BIT 0x14
```

4.3.2.30 GAP_ADTYPE_SIGNED_DATA

```
#define GAP_ADTYPE_SIGNED_DATA 0x13
```

4.3.2.31 GAP_ADTYPE_SIMPLE_PAIRING_HASHC_256

```
#define GAP_ADTYPE_SIMPLE_PAIRING_HASHC_256 0x1D
```

4.3.2.32 GAP_ADTYPE_SIMPLE_PAIRING_RANDR_256

```
#define GAP_ADTYPE_SIMPLE_PAIRING_RANDR_256 0x1E
```

4.3.2.33 GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE

```
#define GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE 0x12
```

4.3.2.34 GAP_ADTYPE_SM_OOB_FLAG

```
#define GAP_ADTYPE_SM_OOB_FLAG 0x11
```

4.3.2.35 GAP_ADTYPE_SM_TK

```
#define GAP_ADTYPE_SM_TK 0x10
```

4.3.2.36 GAP_PUBLIC_ADDR

```
#define GAP_PUBLIC_ADDR 0
```

4.3.2.37 GAP_RAND_ADDR_NRPA

```
#define GAP_RAND_ADDR_NRPA 2
```

4.3.2.38 GAP_RAND_ADDR_RPA

```
#define GAP_RAND_ADDR_RPA 3
```

4.3.2.39 GAP_RAND_ADDR_STATIC

```
#define GAP_RAND_ADDR_STATIC 1
```

4.3.2.40 GAP_SCAN_TYPE_ACTIVE

```
#define GAP_SCAN_TYPE_ACTIVE 1
```

4.3.2.41 GAP_SCAN_TYPE_PASSIVE

```
#define GAP_SCAN_TYPE_PASSIVE 0
```

4.3.2.42 GAP_TX_PWR_CURR_VAL

```
#define GAP_TX_PWR_CURR_VAL 0
```

4.3.2.43 GAP_TX_PWR_MAX_VAL

```
#define GAP_TX_PWR_MAX_VAL 1
```

4.3.2.44 GAPBOND_IO_CAP_DISPLAY_ONLY

```
#define GAPBOND_IO_CAP_DISPLAY_ONLY 0x00
```

4.3.2.45 GAPBOND_IO_CAP_DISPLAY_YES_NO

```
#define GAPBOND_IO_CAP_DISPLAY_YES_NO 0x01
```

4.3.2.46 GAPBOND_IO_CAP_KEYBOARD_DISPLAY

```
#define GAPBOND_IO_CAP_KEYBOARD_DISPLAY 0x04
```

4.3.2.47 GAPBOND_IO_CAP_KEYBOARD_ONLY

```
#define GAPBOND_IO_CAP_KEYBOARD_ONLY 0x02
```

4.3.2.48 GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT

```
#define GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT 0x03
```

4.3.2.49 GAPBOND_PAIRING_MODE_INITIATE

```
#define GAPBOND_PAIRING_MODE_INITIATE 0x02
```

4.3.2.50 GAPBOND_PAIRING_MODE_NO_PAIRING

```
#define GAPBOND_PAIRING_MODE_NO_PAIRING 0x00
```

4.3.2.51 GAPBOND_PAIRING_MODE_WAIT_FOR_REQ

```
#define GAPBOND_PAIRING_MODE_WAIT_FOR_REQ 0x01
```

4.3.2.52 LE_GAP_ADV_MAX_SIZE

```
#define LE_GAP_ADV_MAX_SIZE 31
```

4.3.3 Function Documentation

4.3.3.1 LeGapAddToResolvingList()

```
LE_ERR_STATE LeGapAddToResolvingList (
    LE_BT_ADDR_T * bt_addr,
    UINT8 * irk )
```

Add device to resolving-list.

Parameters

<i>bt_addr</i>	BT device address.
<i>irk</i>	IRK, Identity Resolving Key

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.2 LeGapAddToWhiteList()

```
LE_ERR_STATE LeGapAddToWhiteList (
    LE_BT_ADDR_T * bt_addr )
```

Add device to whitelist.

Parameters

<i>bt_addr</i>	BT device address.
----------------	--------------------

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.3 LeGapAdvertisingEnable()

```
LE_ERR_STATE LeGapAdvertisingEnable (
    BOOL start )
```

Enable or disable advertising function.

Parameters

<i>start</i>	TRUE is enable, FALSE is disable.
--------------	-----------------------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.4 LeGapCentralConnectReq()

```
LE_ERR_STATE LeGapCentralConnectReq (
    LE_BT_ADDR_T * taddr,
    UINT8 own_addr_type )
```

Central connect request.

Parameters

<i>taddr</i>	advertisers device address.
<i>own_addr_type</i>	owner address type.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.5 LeGapCentralSetDataChannel()

```
LE_ERR_STATE LeGapCentralSetDataChannel (
    UINT8 * ch )
```

Central set data channel.

Parameters

<i>ch</i>	data channel.
-----------	---------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.6 LeGapClearResolvingList()

```
LE_ERR_STATE LeGapClearResolvingList (  
    void )
```

Clear the resolving-list in the controller.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.7 LeGapClearWhiteList()

```
LE_ERR_STATE LeGapClearWhiteList (  
    void )
```

Clear whitelist in the controller.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.8 LeGapConnectCancelReq()

```
LE_ERR_STATE LeGapConnectCancelReq (  
    void )
```

Cancel connect request.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.9 LeGapConnParaRequestRsp()

```
void LeGapConnParaRequestRsp (  
    UINT16 conn_hdl,  
    BOOL accept )
```

Connection parameters request response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>accept</i>	TRUE is accept, FALSE is not.

Returns

None.

4.3.3.10 LeGapConnUpdateRequest()

```
void LeGapConnUpdateRequest (
    UINT16 conn_hdl,
    LE_CONN_PARA_T * para )
```

Connection parameters update request.

Parameters

<i>conn_hdl</i>	connection handle.
<i>para</i>	update connection parameters.

Returns

None.

4.3.3.11 LeGapConnUpdateResponse()

```
void LeGapConnUpdateResponse (
    UINT16 conn_hdl,
    UINT8 identifier,
    BOOL accept )
```

Connection parameters update response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>identifier</i>	TBD
<i>accept</i>	accept request, or not.

Returns

None.

4.3.3.12 LeGapDisconnectReq()

```
LE_ERR_STATE LeGapDisconnectReq (
    UINT16 conn_hdl )
```

Disconnect the physical connection.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.13 LeGapGenRandAddr()

```
LE_ERR_STATE LeGapGenRandAddr (
    UINT8 type,
    BD_ADDR addr )
```

Called to generation random address.

Parameters

<i>type</i>	address type.
<i>addr</i>	address.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.14 LeGapGetBtAddr()

```
void LeGapGetBtAddr (
    void )
```

Get owner device address.

4.3.3.15 LeGapReadAdvChannelTxPower()

```
void LeGapReadAdvChannelTxPower (
    void )
```

Read ADV channel txpower.

4.3.3.16 LeGapReadChannelMap()

```
LE_ERR_STATE LeGapReadChannelMap (
    UINT16 conn_hdl )
```

Read channel map.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.17 LeGapReadResolvingListSize()

```
void LeGapReadResolvingListSize (
    void )
```

Read the resolving-list size in the controller.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.18 LeGapReadRssi()

```
LE_ERR_STATE LeGapReadRssi (
    UINT16 conn_hdl )
```

Read RSSI value from controller.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.19 LeGapReadTxPower()

```
LE_ERR_STATE LeGapReadTxPower (
    UINT16 conn_hdl,
    UINT8 type )
```

Read tx power value for the specified connection.

Parameters

<i>conn_hdl</i>	connection handle.
<i>type</i>	current tx power, or maximum tx power. Don't support.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.20 LeGapReadWhiteListSize()

```
void LeGapReadWhiteListSize (
    void )
```

Read whitelist size in the controller.

4.3.3.21 LeGapRemoveFromWhiteList()

```
LE_ERR_STATE LeGapRemoveFromWhiteList (
    LE_BT_ADDR_T * bt_addr )
```

Remove device from whitelist.

Remove device from resolving-list.

Parameters

<i>bt_addr</i>	BT device address.
----------------	--------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.22 LeGapScanningReq()

```
LE_ERR_STATE LeGapScanningReq (  
    BOOL start,  
    BOOL filter )
```

Request scanning start.

Parameters

<i>start</i>	TRUE is start, FALSE is not.
<i>filter</i>	scan policy, refer to <code>LE_HCI_SCAN_FILT_*</code> in ble_hci_if.h

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.23 LeGapSetAdvData()

```
LE_ERR_STATE LeGapSetAdvData (  
    UINT8 len,  
    UINT8 * data )
```

Called to set ADV data.

Parameters

<i>len</i>	ADV data length.
<i>data</i>	ADV data.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.24 LeGapSetAdvParameter()

```
LE_ERR_STATE LeGapSetAdvParameter (
    LE_GAP_ADVERTISING_PARAM_T * params )
```

Called to set ADV parameters.

Parameters

<i>params</i>	advertising params.
---------------	---------------------

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.25 LeGapSetConnParameter()

```
LE_ERR_STATE LeGapSetConnParameter (
    UINT16 interval_min,
    UINT16 interval_max,
    UINT16 slave_latency,
    UINT16 supervision_timeout )
```

Called to set connection parameters.

Parameters

<i>interval_min</i>	mininum connection interval.
<i>interval_max</i>	maxinum connection interval.
<i>slave_latency</i>	slave letency.
<i>supervision_timeout</i>	supervison timeout.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.26 LeGapSetDataChannelPduLen()

```
LE_ERR_STATE LeGapSetDataChannelPduLen (
    UINT16 conn_hdl,
```

```
UINT16 tx_octets,  
UINT16 tx_time )
```

Set data channel PDU length.

Parameters

<i>tx_octets</i>	the maximum number of octets in the Payload field that the local device will send to the remote device.
<i>tx_time</i>	the maximum number of microseconds that the local device will take to transmit a PDU to the remote device.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.27 LeGapSetRandAddr()

```
LE_ERR_STATE LeGapSetRandAddr (  
    BD_ADDR addr )
```

Called to set random address.

Parameters

<i>addr</i>	the random address which should be set.
-------------	---

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.28 LeGapSetRpaTimeout()

```
LE_ERR_STATE LeGapSetRpaTimeout (  
    UINT16 timeout )
```

Set resolvable private address timeout.

Parameters

<i>timeout</i>	RPA_Timeout, measured in seconds.
----------------	-----------------------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.29 LeGapSetStaticAddr()

```
LE_ERR_STATE LeGapSetStaticAddr (
    BD_ADDR addr )
```

Called to set static address.

Parameters

<i>addr</i>	the static address which should be set.
-------------	---

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.30 LeSetScanParameter()

```
LE_ERR_STATE LeSetScanParameter (
    LE_GAP_SCAN_PARAM_T * params )
```

Called to set scan parameters.

Parameters

<i>params</i>	scan parameters.
---------------	------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.3.3.31 LeSetScanRspData()

```
LE_ERR_STATE LeSetScanRspData (
    UINT8 len,
    UINT8 * data )
```

Called to set scan response data.

Parameters

<i>len</i>	scan response data length.
<i>data</i>	scan response data.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4 BLE GATT APIs

Data Structures

- struct [LE_GATT_ATTR_T](#)
- struct [LE_GATT_MSG_ACCESS_READ_IND_T](#)
- struct [LE_GATT_MSG_ACCESS_WRITE_IND_T](#)
- struct [LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T](#)
- struct [LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T](#)
- struct [LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T](#)
- struct [LE_GATT_MSG_CONFIRMATION_CFM_T](#)
- struct [LE_GATT_MSG_EXCHANGE_MTU_CFM_T](#)
- struct [LE_GATT_MSG_EXCHANGE_MTU_IND_T](#)
- struct [LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T](#)
- struct [LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T](#)
- struct [LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T](#)
- struct [LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T](#)
- struct [LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T](#)
- struct [LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T](#)
- struct [LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T](#)
- struct [LE_GATT_MSG_INDICATE_IND_T](#)
- struct [LE_GATT_MSG_NOTIFY_CFM_T](#)
- struct [LE_GATT_MSG_NOTIFY_IND_T](#)
- struct [LE_GATT_MSG_OPERATION_TIMEOUT_T](#)
- struct [LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T](#)
- struct [LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T](#)
- struct [LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T](#)
- struct [LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T](#)
- struct [LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T](#)
- struct [LE_GATT_MSG_SERVICE_INFO_IND_T](#)
- struct [LE_GATT_MSG_SIGNED_WRITE_CFM_T](#)
- struct [LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T](#)
- struct [LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T](#)
- struct [LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T](#)
- struct [LE_GATT_MSG_WRITE_NO_RSP_CFM_T](#)
- struct [LE_GATT_SERVICE_T](#)

Macros

- [#define CHAR_AGGREGATE_DESCRIPTOR](#)(len, pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharAggregateUuid, LE_GATT_PERMIT_READ, 0, len, (UINT8 *) (pVal)}
- [#define CHAR_CLIENT_CONFIG_DESCRIPTOR](#)(permit, pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcClientCharConfigUuid, LE_GATT_PERMIT_READ | permit, 0, 2, (UINT8 *) (pVal)}
- [#define CHAR_DECL_UUID16_ATTR_VAL](#)(prop, type) {(prop), 0, 0, UINT16_LO(type), UINT16_HI(type)}
- [#define CHAR_EXT_PROP_DESCRIPTOR](#)(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharExtPropUuid, LE_GATT_PERMIT_READ, 0, 2, (UINT8 *) (pVal)}
- [#define CHAR_PRESENT_FORMAT_DESCRIPTOR](#)(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharFormatUuid, LE_GATT_PERMIT_READ, 0, 7, (UINT8 *) (pVal)}
- [#define CHAR_SERVER_CONFIG_DESCRIPTOR](#)(permit, pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcServerCharConfigUuid, LE_GATT_PERMIT_READ | permit, 0, 2, (UINT8 *) (pVal)}
- [#define CHAR_USER_DESC_DESCRIPTOR](#)(permit, maxLen, len, pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharUserDescUuid, permit, maxLen, len, (UINT8 *) (pVal)}

- `#define CHARACTERISTIC_DECL_UUID128(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharacteristicUuid, LE_GATT_PERMIT_READ, 0, 19, (UINT8 *) (pVal)}`
- `#define CHARACTERISTIC_DECL_UUID16(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcCharacteristicUuid, LE_GATT_PERMIT_READ, 0, 5, (UINT8 *) (pVal)}`
- `#define CHARACTERISTIC_UUID128(pUuid, permit, maxLen, len, pVal) {0, LE_GATT_UUID128, (UINT16 *) pUuid, permit, maxLen, len, (UINT8 *) (pVal)}`
- `#define CHARACTERISTIC_UUID16(pUuid, permit, maxLen, len, pVal) {0, LE_GATT_UUID16, (UINT16 *) pUuid, permit, maxLen, len, (UINT8 *) (pVal)}`
- `#define GATT_CHAR_AGG_FORMAT_UUID 0x2905`
- `#define GATT_CHAR_EXT_PROPS_UUID 0x2900`
- `#define GATT_CHAR_FORMAT_UUID 0x2904`
- `#define GATT_CHAR_USER_DESC_UUID 0x2901`
- `#define GATT_CHARACTERISTIC_UUID 0x2803`
- `#define GATT_CLIENT_CHAR_CFG_UUID 0x2902`
- `#define GATT_EXT_REPORT_REF_UUID 0x2907`
- `#define GATT_INCLUDE_UUID 0x2802`
- `#define GATT_PRIMARY_SERVICE_UUID 0x2800`
- `#define GATT_REPORT_REF_UUID 0x2908`
- `#define GATT_SECONDARY_SERVICE_UUID 0x2801`
- `#define GATT_SERV_CHAR_CFG_UUID 0x2903`
- `#define GATT_VALID_RANGE_UUID 0x2906`
- `#define INCLUDE_DECL_UUID128(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcIncludeUuid, LE_GATT_PERMIT_READ, 0, 4, (UINT8 *) (pVal)}`
- `#define INCLUDE_DECL_UUID128_ATTR_VAL() {0, 0, 0, 0}`
- `#define INCLUDE_DECL_UUID16_ATTR_VAL(uuid) {0, 0, 0, 0, UINT16_LO(uuid), UINT16_HI(uuid)}`
- `#define INCLUDE_DECL_UUID16(pVal) {0, LE_GATT_UUID16, (UINT16 *)&gcIncludeUuid, LE_GATT_PERMIT_READ, 0, 6, (UINT8 *) (pVal)}`
- `#define LE_ATT_UUID_SIZE 2`
- `#define LE_GATT_CHAR_PROP_AUTH 0x40`
- `#define LE_GATT_CHAR_PROP_BCAST 0x01`

Characteristic Properties Bit.

- `#define LE_GATT_CHAR_PROP_EXT_PROP 0x80`
- `#define LE_GATT_CHAR_PROP_IND 0x20`
- `#define LE_GATT_CHAR_PROP_NTF 0x10`
- `#define LE_GATT_CHAR_PROP_RD 0x02`
- `#define LE_GATT_CHAR_PROP_WR 0x08`
- `#define LE_GATT_CHAR_PROP_WR_NO_RESP 0x04`
- `#define LE_GATT_CLIENT_CFG_INDICATION 0x02`
- `#define LE_GATT_CLIENT_CFG_NOTIFICATION 0x01`
- `#define LE_GATT_EXT_PROP_RELIABLE_WR 0x0001`
- `#define LE_GATT_EXT_PROP_WR_AUX 0x0002`
- `#define LE_GATT_FLAG_PREPARE_WRITE 0x02`
- `#define LE_GATT_FLAG_WRITE_CMD 0x01`
- `#define LE_GATT_FLAG_WRITE_REQ 0x00`
- `#define LE_GATT_PERM_AUTH_READABLE (0x1 << 4)`
- `#define LE_GATT_PERM_AUTH_WRITABLE (0x1 << 6)`
- `#define LE_GATT_PERM_NONE (0x00)`
- `#define LE_GATT_PERM_READ (0x1 << 1)`
- `#define LE_GATT_PERM_RELIABLE_WRITE (0x1 << 5)`
- `#define LE_GATT_PERM_WRITE_CMD (0x1 << 2)`
- `#define LE_GATT_PERM_WRITE_REQ (0x1 << 3)`
- `#define LE_GATT_PERMIT_AUTHEN_READ (0x0040)`
- `#define LE_GATT_PERMIT_AUTHEN_WRITE (0x0080)`
- `#define LE_GATT_PERMIT_AUTHOR_READ (0x0004)`
- `#define LE_GATT_PERMIT_AUTHOR_WRITE (0x0008)`

- `#define LE_GATT_PERMIT_ENCRYPT_READ (0x0010)`
- `#define LE_GATT_PERMIT_ENCRYPT_WRITE (0x0020)`
- `#define LE_GATT_PERMIT_READ (0x0001)`
- `#define LE_GATT_PERMIT_READABLE (LE_GATT_PERMIT_READ | LE_GATT_PERMIT_AUTHEN_READ | LE_GATT_PERMIT_AUTHOR_READ | LE_GATT_PERMIT_ENCRYPT_READ | LE_GATT_PERMIT_SC_AUTHEN_READ)`
- `#define LE_GATT_PERMIT_SC_AUTHEN_READ (0x0100)`
- `#define LE_GATT_PERMIT_SC_AUTHEN_WRITE (0x0200)`
- `#define LE_GATT_PERMIT_WRITABLE (LE_GATT_PERMIT_WRITE | LE_GATT_PERMIT_AUTHEN_WRITE | LE_GATT_PERMIT_AUTHOR_WRITE | LE_GATT_PERMIT_ENCRYPT_WRITE | LE_GATT_PERMIT_SC_AUTHEN_WRITE)`
- `#define LE_GATT_PERMIT_WRITE (0x0002)`
- `#define PRIMARY_SERVICE_DECL_UUID128(pUuid) {0, LE_GATT_UUID16, (UINT16 *)&gcPrimaryServiceUuid, LE_GATT_PERMIT_READ, 0, 16, (UINT8 *)&(pUuid)}`
- `#define PRIMARY_SERVICE_DECL_UUID16(pUuid) {0, LE_GATT_UUID16, (UINT16 *)&gcPrimaryServiceUuid, LE_GATT_PERMIT_READ, 0, 2, (UINT8 *)&(pUuid)}`
- `#define SECONDARY_SERVICE_DECL_UUID128(pUuid) {0, LE_GATT_UUID16, (UINT16 *)&gcSecondaryServiceUuid, LE_GATT_PERMIT_READ, 0, 16, (UINT8 *)&(pUuid)}`
- `#define SECONDARY_SERVICE_DECL_UUID16(pUuid) {0, LE_GATT_UUID16, (UINT16 *)&gcSecondaryServiceUuid, LE_GATT_PERMIT_READ, 0, 2, (UINT8 *)&(pUuid)}`

Enumerations

- `enum {`
`LE_GATT_MSG_INIT_CFM = LE_GATT_MSG_BASE, LE_GATT_MSG_EXCHANGE_MTU_IND,`
`LE_GATT_MSG_EXCHANGE_MTU_CFM,`
`LE_GATT_MSG_ACCESS_READ_IND,`
`LE_GATT_MSG_ACCESS_WRITE_IND, LE_GATT_MSG_SERVICE_INFO_IND,`
`LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM,`
`LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM,`
`LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM, LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND,`
`LE_GATT_MSG_FIND_CHARACTERISTIC_CFM, LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND,`
`LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM, LE_GATT_MSG_CHARACTERISTIC_VAL_IND,`
`LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM, LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM,`
`LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM, LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM,`
`LE_GATT_MSG_WRITE_CHAR_VALUE_CFM, LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM,`
`LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM, LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM,`
`LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM, LE_GATT_MSG_WRITE_NO_RSP_CFM,`
`LE_GATT_MSG_SIGNED_WRITE_CFM, LE_GATT_MSG_NOTIFY_IND, LE_GATT_MSG_NOTIFY_CFM,`
`LE_GATT_MSG_INDICATE_IND,`
`LE_GATT_MSG_CONFIRMATION_CFM, LE_GATT_MSG_OPERATION_TIMEOUT,`
`LE_GATT_MSG_SIGN_RESOLUTION_FAIL,`
`LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND,`
`LE_GATT_MSG_TOP };`

BLE GATT message id.

Functions

- `LE_ERR_STATE LeGattAccessReadRsp (UINT16 conn_hdl, UINT16 handle, UINT8 att_err)`
Gatt access read response.
- `LE_ERR_STATE LeGattAccessWriteRsp (UINT16 conn_hdl, UINT8 method, UINT16 handle, UINT8 att_err)`
Gatt access write response.
- `LE_ERR_STATE LeGattChangeAttrVal (LE_GATT_SERVICE_T *svc, UINT16 attrId, UINT16 len, void *val)`
Change attribute value.
- `LE_ERR_STATE LeGattCharValConfirmation (UINT16 conn_hdl)`
Prepare write characteristic value response.
- `LE_ERR_STATE LeGattCharValIndicate (UINT16 conn_hdl, UINT16 hdl, UINT16 len, UINT8 *pval)`
Gatt characteristic value indication.

- LE_ERR_STATE [LeGattCharValNotify](#) (UINT16 conn_hdl, UINT16 hdl, UINT16 len, UINT8 *pval)
Gatt characteristic value notification.
- LE_ERR_STATE [LeGattExchangeMtuReq](#) (UINT16 conn_hdl, UINT16 mtu)
Exchange MTU request.
- LE_ERR_STATE [LeGattExchangeMtuRsp](#) (UINT16 conn_hdl, UINT16 mtu)
Exchange MTU response.
- LE_ERR_STATE [LeGattExecuteWriteCharValReliable](#) (UINT16 conn_hdl, BOOL yesno)
Execute write characteristic value request.
- LE_ERR_STATE [LeGattFindAllCharacteristic](#) (UINT16 conn_hdl, UINT16 start_hdl, UINT16 end_hdl)
Find all characteristic.
- LE_ERR_STATE [LeGattFindAllCharDescriptor](#) (UINT16 conn_hdl, UINT16 start_hdl, UINT16 end_hdl)
Find all characteristic description.
- LE_ERR_STATE [LeGattFindAllPrimaryService](#) (UINT16 conn_hdl)
Find all primary service.
- LE_ERR_STATE [LeGattFindCharacteristicByUuid](#) (UINT16 conn_hdl, UINT16 start_hdl, UINT16 end_hdl, UINT8 format, UINT16 *uuid)
Find characteristic by UUID.
- LE_ERR_STATE [LeGattFindIncludedService](#) (UINT16 conn_hdl, UINT16 start_hdl, UINT16 end_hdl)
Find include service.
- LE_ERR_STATE [LeGattFindPrimaryServiceByUuid](#) (UINT16 conn_hdl, UINT8 format, UINT16 *uuid)
Find primary service by UUID.
- UINT16 [LeGattGetAttrHandle](#) (LE_GATT_SERVICE_T *svc, UINT16 attrId)
Get attribute handle.
- LE_ERR_STATE [LeGattGetAttrVal](#) (LE_GATT_SERVICE_T *svc, UINT16 attrId, UINT16 *len, void *val)
Get attribute value.
- UINT16 [LeGattGetAttrValLen](#) (LE_GATT_SERVICE_T *svc, UINT16 attrId)
Get the length of attribute value.
- UINT16 [LeGattGetAttrValMaxLen](#) (LE_GATT_SERVICE_T *svc, UINT16 attrId)
Get the max length of attribute value.
- void [LeGattInit](#) (TASK appTask)
BLE Gatt module init.
- LE_ERR_STATE [LeGattModifyAttrVal](#) (LE_GATT_SERVICE_T *svc, UINT16 attrId, UINT16 offset, UINT16 len, void *val)
Modify attribute value.
- LE_ERR_STATE [LeGattPrepareWriteCharValReliable](#) (UINT16 conn_hdl, UINT16 handle, UINT16 offset, UINT16 len, UINT8 *val)
Prepare write characteristic value request.
- LE_ERR_STATE [LeGattReadCharValByUuid](#) (UINT16 conn_hdl, UINT16 start_hdl, UINT16 end_hdl, UINT8 format, UINT16 *uuid)
Read a characteristic value by UUID.
- LE_ERR_STATE [LeGattReadCharValue](#) (UINT16 conn_hdl, UINT16 handle)
Read a characteristic value.
- LE_ERR_STATE [LeGattReadLongCharVal](#) (UINT16 conn_hdl, UINT16 handle, UINT16 offset)
Read a long characteristic value.
- LE_ERR_STATE [LeGattReadMultipleCharVal](#) (UINT16 conn_hdl, UINT16 count, UINT16 *handle)
Read Multiple characteristic values.
- LE_ERR_STATE [LeGattRegisterIncludeService](#) (UINT16 inc_hdl, UINT16 start_hdl, UINT16 end_hdl, UI↔NT16 uuid)
Called to register an include service.
- LE_GATT_SERVICE_T * [LeGattRegisterService](#) (LE_GATT_ATTR_T *attrTable, UINT16 numAttr)
Called to register a service.

- LE_ERR_STATE [LeGattSignedWriteNoRsp](#) (UINT16 conn_hdl, UINT16 handle, UINT16 len, UINT8 *val)
Signed write without response.
- void [LeGattStopCurrentProcedure](#) (UINT16 conn_hdl)
Stop current procedure.
- LE_ERR_STATE [LeGattWriteCharVal](#) (UINT16 conn_hdl, UINT16 handle, UINT16 len, UINT8 *val)
Write characteristic value.
- LE_ERR_STATE [LeGattWriteCharValReliable](#) (UINT16 conn_hdl, UINT16 handle, UINT16 offset, UINT16 len, UINT8 *val)
Write characteristic value reliable.
- LE_ERR_STATE [LeGattWriteLongCharVal](#) (UINT16 conn_hdl, UINT16 handle, UINT16 offset, UINT16 len, UINT8 *val)
Write long characteristic value.
- LE_ERR_STATE [LeGattWriteNoRsp](#) (UINT16 conn_hdl, UINT16 handle, UINT16 len, UINT8 *val)
Write without response.

Variables

- const UINT16 [gcCharacteristicUuid](#)
- const UINT16 [gcCharAggregateUuid](#)
- const UINT16 [gcCharExtPropUuid](#)
- const UINT16 [gcCharFormatUuid](#)
- const UINT16 [gcCharUserDescUuid](#)
- const UINT16 [gcClientCharConfigUuid](#)
- const UINT16 [gcExtReportRefUuid](#)
- const UINT16 [gcIncludeUuid](#)
- const UINT16 [gcPrimaryServiceUuid](#)
- const UINT16 [gcReportRefUuid](#)
- const UINT16 [gcSecondaryServiceUuid](#)
- const UINT16 [gcServerCharConfigUuid](#)
- const UINT16 [gcValidRangeUuid](#)

4.4.1 Detailed Description

4.4.2 Macro Definition Documentation

4.4.2.1 CHARAggregateDescriptor

```
#define CHARAggregateDescriptor(  
    len,  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharAggregateUuid, LE_GATT_PERMIT_READ,  
0, len, (UINT8 *) (pVal)}
```

4.4.2.2 CHAR_CLIENT_CONFIG_DESCRIPTOR

```
#define CHAR_CLIENT_CONFIG_DESCRIPTOR(  
    permit,  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcClientCharConfigUuid, LE_GATT_PERMIT_READ  
| permit, 0, 2, (UINT8 *) (pVal)}
```

4.4.2.3 CHAR_DECL_UUID16_ATTR_VAL

```
#define CHAR_DECL_UUID16_ATTR_VAL(  
    prop,  
    type ) {(prop), 0, 0, UINT16_LO(type), UINT16_HI(type)}
```

4.4.2.4 CHAR_EXT_PROP_DESCRIPTOR

```
#define CHAR_EXT_PROP_DESCRIPTOR(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharExtPropUuid, LE_GATT_PERMIT_READ, 0,  
2, (UINT8 *) (pVal)}
```

4.4.2.5 CHAR_PRESENT_FORMAT_DESCRIPTOR

```
#define CHAR_PRESENT_FORMAT_DESCRIPTOR(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharFormatUuid, LE_GATT_PERMIT_READ, 0,  
7, (UINT8 *) (pVal)}
```

4.4.2.6 CHAR_SERVER_CONFIG_DESCRIPTOR

```
#define CHAR_SERVER_CONFIG_DESCRIPTOR(  
    permit,  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcServerCharConfigUuid, LE_GATT_PERMIT_READ  
| permit, 0, 2, (UINT8 *) (pVal)}
```

4.4.2.7 CHAR_USER_DESC_DESCRIPTOR

```
#define CHAR_USER_DESC_DESCRIPTOR(  
    permit,  
    maxLen,  
    len,  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharUserDescUuid, permit, maxLen, len,  
(UINT8 *) (pVal)}
```


4.4.2.8 CHARACTERISTIC_DECL_UUID128

```
#define CHARACTERISTIC_DECL_UUID128(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharacteristicUuid, LE_GATT_PERMIT_READ,  
0, 19, (UINT8 *) (pVal)}
```

4.4.2.9 CHARACTERISTIC_DECL_UUID16

```
#define CHARACTERISTIC_DECL_UUID16(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcCharacteristicUuid, LE_GATT_PERMIT_READ,  
0, 5, (UINT8 *) (pVal)}
```

4.4.2.10 CHARACTERISTIC_UUID128

```
#define CHARACTERISTIC_UUID128(  
    pUuid,  
    permit,  
    maxLen,  
    len,  
    pVal ) {0, LE_GATT_UUID128, (UINT16 *)pUuid, permit, maxLen, len, (UINT8 *) (pVal)}
```

4.4.2.11 CHARACTERISTIC_UUID16

```
#define CHARACTERISTIC_UUID16(  
    pUuid,  
    permit,  
    maxLen,  
    len,  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)pUuid, permit, maxLen, len, (UINT8 *) (pVal)}
```

4.4.2.12 GATT_CHAR_AGG_FORMAT_UUID

```
#define GATT_CHAR_AGG_FORMAT_UUID 0x2905
```

4.4.2.13 GATT_CHAR_EXT_PROPS_UUID

```
#define GATT_CHAR_EXT_PROPS_UUID 0x2900
```

4.4.2.14 GATT_CHAR_FORMAT_UUID

```
#define GATT_CHAR_FORMAT_UUID 0x2904
```

4.4.2.15 GATT_CHAR_USER_DESC_UUID

```
#define GATT_CHAR_USER_DESC_UUID 0x2901
```

4.4.2.16 GATT_CHARACTERISTIC_UUID

```
#define GATT_CHARACTERISTIC_UUID 0x2803
```

4.4.2.17 GATT_CLIENT_CHAR_CFG_UUID

```
#define GATT_CLIENT_CHAR_CFG_UUID 0x2902
```

4.4.2.18 GATT_EXT_REPORT_REF_UUID

```
#define GATT_EXT_REPORT_REF_UUID 0x2907
```

4.4.2.19 GATT_INCLUDE_UUID

```
#define GATT_INCLUDE_UUID 0x2802
```

4.4.2.20 GATT_PRIMARY_SERVICE_UUID

```
#define GATT_PRIMARY_SERVICE_UUID 0x2800
```

4.4.2.21 GATT_REPORT_REF_UUID

```
#define GATT_REPORT_REF_UUID 0x2908
```

4.4.2.22 GATT_SECONDARY_SERVICE_UUID

```
#define GATT_SECONDARY_SERVICE_UUID 0x2801
```

4.4.2.23 GATT_SERV_CHAR_CFG_UUID

```
#define GATT_SERV_CHAR_CFG_UUID 0x2903
```

4.4.2.24 GATT_VALID_RANGE_UUID

```
#define GATT_VALID_RANGE_UUID 0x2906
```

4.4.2.25 INCLUDE_DECL_UUID128

```
#define INCLUDE_DECL_UUID128(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcIncludeUuid, LE_GATT_PERMIT_READ, 0, 4,  
(UINT8 *) (pVal)}
```

4.4.2.26 INCLUDE_DECL_UUID128_ATTR_VAL

```
#define INCLUDE_DECL_UUID128_ATTR_VAL( ) {0, 0, 0, 0}
```

4.4.2.27 INCLUDE_DECL_UUID16_ATTR_VAL

```
#define INCLUDE_DECL_UUID16_ATTR_VAL(  
    uuid ) {0, 0, 0, 0, UINT16_LO(uuid), UINT16_HI(uuid)}
```

4.4.2.28 INCLUDE_DECL_UUIN16

```
#define INCLUDE_DECL_UUIN16(  
    pVal ) {0, LE_GATT_UUID16, (UINT16 *)&gcIncludeUuid, LE_GATT_PERMIT_READ, 0, 6,  
(UINT8 *) (pVal)}
```

4.4.2.29 LE_ATT_UUID_SIZE

```
#define LE_ATT_UUID_SIZE 2
```

4.4.2.30 LE_GATT_CHAR_PROP_AUTH

```
#define LE_GATT_CHAR_PROP_AUTH 0x40
```

4.4.2.31 LE_GATT_CHAR_PROP_BCAST

```
#define LE_GATT_CHAR_PROP_BCAST 0x01
```

Characteristic Properties Bit.

4.4.2.32 LE_GATT_CHAR_PROP_EXT_PROP

```
#define LE_GATT_CHAR_PROP_EXT_PROP 0x80
```

4.4.2.33 LE_GATT_CHAR_PROP_IND

```
#define LE_GATT_CHAR_PROP_IND 0x20
```

4.4.2.34 LE_GATT_CHAR_PROP_NTF

```
#define LE_GATT_CHAR_PROP_NTF 0x10
```

4.4.2.35 LE_GATT_CHAR_PROP_RD

```
#define LE_GATT_CHAR_PROP_RD 0x02
```

4.4.2.36 LE_GATT_CHAR_PROP_WR

```
#define LE_GATT_CHAR_PROP_WR 0x08
```

4.4.2.37 LE_GATT_CHAR_PROP_WR_NO_RESP

```
#define LE_GATT_CHAR_PROP_WR_NO_RESP 0x04
```

4.4.2.38 LE_GATT_CLIENT_CFG_INDICATION

```
#define LE_GATT_CLIENT_CFG_INDICATION 0x02
```

4.4.2.39 LE_GATT_CLIENT_CFG_NOTIFICATION

```
#define LE_GATT_CLIENT_CFG_NOTIFICATION 0x01
```

4.4.2.40 LE_GATT_EXT_PROP_RELIABLE_WR

```
#define LE_GATT_EXT_PROP_RELIABLE_WR 0x0001
```

4.4.2.41 LE_GATT_EXT_PROP_WR_AUX

```
#define LE_GATT_EXT_PROP_WR_AUX 0x0002
```

4.4.2.42 LE_GATT_FLAG_PREPARE_WRITE

```
#define LE_GATT_FLAG_PREPARE_WRITE 0x02
```

4.4.2.43 LE_GATT_FLAG_WRITE_CMD

```
#define LE_GATT_FLAG_WRITE_CMD 0x01
```

4.4.2.44 LE_GATT_FLAG_WRITE_REQ

```
#define LE_GATT_FLAG_WRITE_REQ 0x00
```

4.4.2.45 LE_GATT_PERM_AUTH_READABLE

```
#define LE_GATT_PERM_AUTH_READABLE (0x1<<4)
```

4.4.2.46 LE_GATT_PERM_AUTH_WRITABLE

```
#define LE_GATT_PERM_AUTH_WRITABLE (0x1<<6)
```

4.4.2.47 LE_GATT_PERM_NONE

```
#define LE_GATT_PERM_NONE (0x00)
```

4.4.2.48 LE_GATT_PERM_READ

```
#define LE_GATT_PERM_READ (0x1<<1)
```

4.4.2.49 LE_GATT_PERM_RELIABLE_WRITE

```
#define LE_GATT_PERM_RELIABLE_WRITE (0x1<<5)
```

4.4.2.50 LE_GATT_PERM_WRITE_CMD

```
#define LE_GATT_PERM_WRITE_CMD (0x1<<2)
```

4.4.2.51 LE_GATT_PERM_WRITE_REQ

```
#define LE_GATT_PERM_WRITE_REQ (0x1<<3)
```

4.4.2.52 LE_GATT_PERMIT_AUTHEN_READ

```
#define LE_GATT_PERMIT_AUTHEN_READ (0x0040)
```

4.4.2.53 LE_GATT_PERMIT_AUTHEN_WRITE

```
#define LE_GATT_PERMIT_AUTHEN_WRITE (0x0080)
```

4.4.2.54 LE_GATT_PERMIT_AUTHOR_READ

```
#define LE_GATT_PERMIT_AUTHOR_READ (0x0004)
```

4.4.2.55 LE_GATT_PERMIT_AUTHOR_WRITE

```
#define LE_GATT_PERMIT_AUTHOR_WRITE (0x0008)
```

4.4.2.56 LE_GATT_PERMIT_ENCRYPT_READ

```
#define LE_GATT_PERMIT_ENCRYPT_READ (0x0010)
```

4.4.2.57 LE_GATT_PERMIT_ENCRYPT_WRITE

```
#define LE_GATT_PERMIT_ENCRYPT_WRITE (0x0020)
```

4.4.2.58 LE_GATT_PERMIT_READ

```
#define LE_GATT_PERMIT_READ (0x0001)
```

4.4.2.59 LE_GATT_PERMIT_READABLE

```
#define LE_GATT_PERMIT_READABLE (LE_GATT_PERMIT_READ | LE_GATT_PERMIT_AUTHEN_READ |  
LE_GATT_PERMIT_AUTHOR_READ | LE_GATT_PERMIT_ENCRYPT_READ | LE_GATT_PERMIT_SC_AUTHEN_READ)
```

4.4.2.60 LE_GATT_PERMIT_SC_AUTHEN_READ

```
#define LE_GATT_PERMIT_SC_AUTHEN_READ (0x0100)
```

4.4.2.61 LE_GATT_PERMIT_SC_AUTHEN_WRITE

```
#define LE_GATT_PERMIT_SC_AUTHEN_WRITE (0x0200)
```

4.4.2.62 LE_GATT_PERMIT_WRITABLE

```
#define LE_GATT_PERMIT_WRITABLE (LE_GATT_PERMIT_WRITE | LE_GATT_PERMIT_AUTHEN_WRITE |  
LE_GATT_PERMIT_AUTHOR_WRITE | LE_GATT_PERMIT_ENCRYPT_WRITE | LE_GATT_PERMIT_SC_AUTHEN_WRITE)
```

4.4.2.63 LE_GATT_PERMIT_WRITE

```
#define LE_GATT_PERMIT_WRITE (0x0002)
```

4.4.2.64 PRIMARY_SERVICE_DECL_UUID128

```
#define PRIMARY_SERVICE_DECL_UUID128(  
    pUuid ) {0, LE_GATT_UUID16, (UINT16 *)&gcPrimaryServiceUuid, LE_GATT_PERMIT_READ,  
0, 16, (UINT8 *) (pUuid)}
```

4.4.2.65 PRIMARY_SERVICE_DECL_UUID16

```
#define PRIMARY_SERVICE_DECL_UUID16(  
    pUuid ) {0, LE_GATT_UUID16, (UINT16 *)&gcPrimaryServiceUuid, LE_GATT_PERMIT_READ,  
0, 2, (UINT8 *) (pUuid)}
```

4.4.2.66 SECONDARY_SERVICE_DECL_UUID128

```
#define SECONDARY_SERVICE_DECL_UUID128(  
    pUuid ) {0, LE_GATT_UUID16, (UINT16 *)&gcSecondaryServiceUuid, LE_GATT_PERMIT_READ,  
0, 16, (UINT8 *) (pUuid)}
```


4.4.2.67 SECONDARY_SERVICE_DECL_UUID16

```
#define SECONDARY_SERVICE_DECL_UUID16(  
    pUuid ) {0, LE_GATT_UUID16, (UINT16 *)&gcSecondaryServiceUuid, LE_GATT_PERMIT_READ,  
    0, 2, (UINT8 *) (pUuid)}
```

4.4.3 Enumeration Type Documentation

4.4.3.1 anonymous enum

anonymous enum

BLE GATT message id.

Enumerator

LE_GATT_MSG_INIT_CFM	initialize confirm message
LE_GATT_MSG_EXCHANGE_MTU_IND	exchange MTU indication
LE_GATT_MSG_EXCHANGE_MTU_CFM	exchange MTU confirm
LE_GATT_MSG_ACCESS_READ_IND	access read indication
LE_GATT_MSG_ACCESS_WRITE_IND	access write indication
LE_GATT_MSG_SERVICE_INFO_IND	service information indication
LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_↔ _CFM	find all primary service confirm
LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_↔ _UUID_CFM	find primary service by UUID confirm
LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM	find include service confirm
LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_↔ _IND	characteristic declaration info indication
LE_GATT_MSG_FIND_CHARACTERISTIC_CFM	find characteristic confirm
LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND	characteristic descriptor info indication
LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM	find all characteristic descriptors confirm
LE_GATT_MSG_CHARACTERISTIC_VAL_IND	characteristic value, indication message
LE_GATT_MSG_READ_CHARACTERISTIC_VAL_↔ _UE_CFM	read characteristic value, confirm message
LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_↔ _CFM	read characteristic value by UUID confirm message
LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM	read long characteristic value confirm message
LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_↔ _CFM	read multiple characteristic value confirm
LE_GATT_MSG_WRITE_CHAR_VALUE_CFM	write characteristic value confirm
LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_↔ _CFM	write long characteristic value confirm
LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_↔ _CFM	write characteristic value reliable confirm
LE_GATT_MSG_PREPARE_WRITE_RELIABLE_↔ _CFM	prepare write reliable confirm
LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_↔ _CFM	execute write reliable confirm

Enumerator

LE_GATT_MSG_WRITE_NO_RSP_CFM	write no response confirm
LE_GATT_MSG_SIGNED_WRITE_CFM	signed write confirm
LE_GATT_MSG_NOTIFY_IND	notify indication
LE_GATT_MSG_NOTIFY_CFM	notify confirm
LE_GATT_MSG_INDICATE_IND	indicate indication
LE_GATT_MSG_CONFIRMATION_CFM	confirmation confirm
LE_GATT_MSG_OPERATION_TIMEOUT	operation timeout
LE_GATT_MSG_SIGN_RESOLUTION_FAIL	sign resolution fail
LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND	include service information
LE_GATT_MSG_TOP	top of GATT message id

4.4.4 Function Documentation

4.4.4.1 LeGattAccessReadRsp()

```
LE_ERR_STATE LeGattAccessReadRsp (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT8 att_err )
```

Gatt access read response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	attribute handle.
<i>att_err</i>	0 is OK, others refer to LE_ATT_ERR_* in ble_att_if.h .

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.2 LeGattAccessWriteRsp()

```
LE_ERR_STATE LeGattAccessWriteRsp (
    UINT16 conn_hdl,
    UINT8 method,
    UINT16 handle,
    UINT8 att_err )
```

Gatt access write response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>method</i>	refer to LE_GATT_FLAG_* in ble_gatt_if.h
<i>handle</i>	attribute handle.
<i>att_err</i>	0 is OK, others refer to LE_ATT_ERR_* in ble_att_if.h .

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.3 LeGattChangeAttrVal()

```
LE_ERR_STATE LeGattChangeAttrVal (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId,
    UINT16 len,
    void * val )
```

Change attribute value.

Parameters

	<i>svc</i>	service.
	<i>attrId</i>	attribute index of service.
in	<i>len</i>	attribute value length.
in	<i>val</i>	attribute value.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.4 LeGattCharValConfirmation()

```
LE_ERR_STATE LeGattCharValConfirmation (
    UINT16 conn_hdl )
```

Prepare write characteristic value response.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.5 LeGattCharValIndicate()

```
LE_ERR_STATE LeGattCharValIndicate (
    UINT16 conn_hdl,
    UINT16 hdl,
    UINT16 len,
    UINT8 * pval )
```

Gatt characteristic value indication.

Parameters

<i>conn_hdl</i>	connection handle.
<i>hdl</i>	characteristic value handle.
<i>len</i>	value length.
<i>pval</i>	value.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.6 LeGattCharValNotify()

```
LE_ERR_STATE LeGattCharValNotify (
    UINT16 conn_hdl,
    UINT16 hdl,
    UINT16 len,
    UINT8 * pval )
```

Gatt characteristic value notification.

Parameters

<i>conn_hdl</i>	connection handle.
<i>hdl</i>	characteristic value handle.
<i>len</i>	value length.
<i>pval</i>	value.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.7 LeGattExchangeMtuReq()

```
LE_ERR_STATE LeGattExchangeMtuReq (
    UINT16 conn_hdl,
    UINT16 mtu )
```

Exchange MTU request.

Parameters

<i>conn_hdl</i>	connection handle.
<i>mtu</i>	MTU.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.8 LeGattExchangeMtuRsp()

```
LE_ERR_STATE LeGattExchangeMtuRsp (
    UINT16 conn_hdl,
    UINT16 mtu )
```

Exchange MTU response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>mtu</i>	MTU.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.9 LeGattExecuteWriteCharValReliable()

```
LE_ERR_STATE LeGattExecuteWriteCharValReliable (
    UINT16 conn_hdl,
    BOOL yesno )
```

Execute write characteristic value request.

Parameters

<i>conn_hdl</i>	connection handle.
<i>yesno</i>	execute write or not.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.10 LeGattFindAllCharacteristic()

```
LE_ERR_STATE LeGattFindAllCharacteristic (
    UINT16 conn_hdl,
    UINT16 start_hdl,
    UINT16 end_hdl )
```

Find all characteristic.

Parameters

<i>conn_hdl</i>	connection handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.11 LeGattFindAllCharDescriptor()

```
LE_ERR_STATE LeGattFindAllCharDescriptor (
    UINT16 conn_hdl,
    UINT16 start_hdl,
    UINT16 end_hdl )
```

Find all characteristic description.

Parameters

<i>conn_hdl</i>	connection handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.12 `LeGattFindAllPrimaryService()`

```
LE_ERR_STATE LeGattFindAllPrimaryService (
    UINT16 conn_hdl )
```

Find all primary service.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.13 `LeGattFindCharacteristicByUuid()`

```
LE_ERR_STATE LeGattFindCharacteristicByUuid (
    UINT16 conn_hdl,
    UINT16 start_hdl,
    UINT16 end_hdl,
    UINT8 format,
    UINT16 * uuid )
```

Find characteristic by UUID.

Parameters

<i>conn_hdl</i>	connection handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.
<i>format</i>	UUID type.
<i>uuid</i>	UUID.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.14 LeGattFindIncludedService()

```
LE_ERR_STATE LeGattFindIncludedService (
    UINT16 conn_hdl,
    UINT16 start_hdl,
    UINT16 end_hdl )
```

Find include service.

Parameters

<i>conn_hdl</i>	connection handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.15 LeGattFindPrimaryServiceByUuid()

```
LE_ERR_STATE LeGattFindPrimaryServiceByUuid (
    UINT16 conn_hdl,
    UINT8 format,
    UINT16 * uuid )
```

Find primary service by UUID.

Parameters

<i>conn_hdl</i>	connection handle.
<i>format</i>	UUID type.
<i>uuid</i>	UUID.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.16 LeGattGetAttrHandle()

```

UINT16 LeGattGetAttrHandle (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId )

```

Get attribute handle.

Parameters

<i>svc</i>	service.
<i>attrId</i>	attribute index of service.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.17 LeGattGetAttrVal()

```

LE_ERR_STATE LeGattGetAttrVal (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId,
    UINT16 * len,
    void * val )

```

Get attribute value.

Parameters

	<i>svc</i>	service.
	<i>attrId</i>	attribute index of service.
out	<i>len</i>	attribute value length.
out	<i>val</i>	attribute value.

Returns

- SYS_ERR_SUCCESS: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.18 LeGattGetAttrValLen()

```

UINT16 LeGattGetAttrValLen (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId )

```

Get the length of attribute value.

Parameters

<i>svc</i>	service.
<i>attr↔ Id</i>	attribute index of service.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.19 `LeGattGetAttrValMaxLen()`

```
UINT16 LeGattGetAttrValMaxLen (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId )
```

Get the max length of attribute value.

Parameters

<i>svc</i>	service.
<i>attr↔ Id</i>	attribute index of service.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.20 `LeGattInit()`

```
void LeGattInit (
    TASK appTask )
```

BLE Gatt module init.

Parameters

<i>appTask</i>	the reference of BLE task.
----------------	----------------------------

Returns

None.

4.4.4.21 LeGattModifyAttrVal()

```
LE_ERR_STATE LeGattModifyAttrVal (
    LE_GATT_SERVICE_T * svc,
    UINT16 attrId,
    UINT16 offset,
    UINT16 len,
    void * val )
```

Modify attribute value.

Parameters

<i>svc</i>	servie.
<i>attrId</i>	attribute index of service.
<i>offset</i>	modify offset.
<i>len</i>	modify length.
<i>val</i>	modify value.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.22 LeGattPrepareWriteCharValReliable()

```
LE_ERR_STATE LeGattPrepareWriteCharValReliable (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 offset,
    UINT16 len,
    UINT8 * val )
```

Prepare write characteristic value request.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>offset</i>	offset written.
<i>len</i>	length written.
<i>val</i>	value.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.23 LeGattReadCharValByUuid()

```
LE_ERR_STATE LeGattReadCharValByUuid (
    UINT16 conn_hdl,
    UINT16 start_hdl,
    UINT16 end_hdl,
    UINT8 format,
    UINT16 * uuid )
```

Read a characteristic value by UUID.

Parameters

<i>conn_hdl</i>	connection handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.
<i>format</i>	UUID type.
<i>uuid</i>	UUID.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.24 LeGattReadCharValue()

```
LE_ERR_STATE LeGattReadCharValue (
    UINT16 conn_hdl,
    UINT16 handle )
```

Read a characteristic value.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.25 LeGattReadLongCharVal()

```
LE_ERR_STATE LeGattReadLongCharVal (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 offset )
```

Read a long characteristic value.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>offset</i>	characteristic value offset.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.26 LeGattReadMultipleCharVal()

```
LE_ERR_STATE LeGattReadMultipleCharVal (
    UINT16 conn_hdl,
    UINT16 count,
    UINT16 * handle )
```

Read Multiple characteristic values.

Parameters

<i>conn_hdl</i>	connection handle.
<i>count</i>	handle count.
<i>handle</i>	handle table.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.27 LeGattRegisterIncludeService()

```
LE_ERR_STATE LeGattRegisterIncludeService (
    UINT16 inc_hdl,
```

```
UINT16 start_hdl,  
UINT16 end_hdl,  
UINT16 uuid )
```

Called to register an include service.

Parameters

<i>inc_hdl</i>	include service handle.
<i>start_hdl</i>	start handle.
<i>end_hdl</i>	end handle.
<i>uuid</i>	include service UUID.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.28 LeGattRegisterService()

```
LE_GATT_SERVICE_T* LeGattRegisterService (  
    LE_GATT_ATTR_T * attrTable,  
    UINT16 numAttr )
```

Called to register a service.

Parameters

<i>attrTable</i>	service attribute table.
<i>numAttr</i>	the attribute number of service.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.29 LeGattSignedWriteNoRsp()

```
LE_ERR_STATE LeGattSignedWriteNoRsp (  
    UINT16 conn_hdl,  
    UINT16 handle,  
    UINT16 len,  
    UINT8 * val )
```

Signed write without response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>len</i>	length of the data to be written.
<i>val</i>	the value to be written.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.30 LeGattStopCurrentProcedure()

```
void LeGattStopCurrentProcedure (
    UINT16 conn_hdl )
```

Stop current procedure.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.31 LeGattWriteCharVal()

```
LE_ERR_STATE LeGattWriteCharVal (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 len,
    UINT8 * val )
```

Write characteristic value.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>len</i>	length of the data to be written.
<i>val</i>	the value to be written.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.32 `LeGattWriteCharValReliable()`

```
LE_ERR_STATE LeGattWriteCharValReliable (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 offset,
    UINT16 len,
    UINT8 * val )
```

Write characteristic value reliable.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>offset</i>	offset written.
<i>len</i>	length written.
<i>val</i>	value.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.33 `LeGattWriteLongCharVal()`

```
LE_ERR_STATE LeGattWriteLongCharVal (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 offset,
    UINT16 len,
    UINT8 * val )
```

Write long characteristic value.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>offset</i>	value position offset.
<i>len</i>	length of the data to be written.
<i>val</i>	the value to be written.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.4.34 LeGattWriteNoRsp()

```
LE_ERR_STATE LeGattWriteNoRsp (
    UINT16 conn_hdl,
    UINT16 handle,
    UINT16 len,
    UINT8 * val )
```

Write without response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>handle</i>	characteristic value handle.
<i>len</i>	length of the data to be written.
<i>val</i>	the value to be written.

Returns

- `SYS_ERR_SUCCESS`: success.
- others: refer to error code in [ble_err.h](#).

4.4.5 Variable Documentation

4.4.5.1 gcCharacteristicUuid

```
const UINT16 gcCharacteristicUuid
```

4.4.5.2 gcCharAggregateUuid

```
const UINT16 gcCharAggregateUuid
```

4.4.5.3 gcCharExtPropUuid

```
const UINT16 gcCharExtPropUuid
```

4.4.5.4 gcCharFormatUuid

```
const UINT16 gcCharFormatUuid
```

4.4.5.5 gcCharUserDescUuid

```
const UINT16 gcCharUserDescUuid
```

4.4.5.6 gcClientCharConfigUuid

```
const UINT16 gcClientCharConfigUuid
```

4.4.5.7 gcExtReportRefUuid

```
const UINT16 gcExtReportRefUuid
```

4.4.5.8 gcIncludeUuid

```
const UINT16 gcIncludeUuid
```

4.4.5.9 gcPrimaryServiceUuid

```
const UINT16 gcPrimaryServiceUuid
```

4.4.5.10 gcReportRefUuid

```
const UINT16 gcReportRefUuid
```

4.4.5.11 gcSecondaryServiceUuid

```
const UINT16 gcSecondaryServiceUuid
```

4.4.5.12 gcServerCharConfigUuid

```
const UINT16 gcServerCharConfigUuid
```

4.4.5.13 gcValidRangeUuid

```
const UINT16 gcValidRangeUuid
```

4.5 BLE MSG APIs

Data Structures

- struct [LE_SYS_MSG_BUF_OVERFLOW_T](#)

Macros

- `#define LE_ATT_MSG_BASE 0x1400`
- `#define LE_CM_MSG_BASE 0x1100`
- `#define LE_GATT_MSG_BASE 0x1500`
- `#define LE_HCI_MSG_BASE 0x1000`
- `#define LE_L2CAP_MSG_BASE 0x1200`
- `#define LE_SMP_MSG_BASE 0x1300`
- `#define LE_SYS_MSG_BASE 0x8000`
- `#define MESSAGE_ALLOCATE(M, S) PanicUnlessMalloc(sizeof(M##_T) + S)`
- `#define MESSAGE_BULID(M) M##_T *msg = PanicUnlessMalloc(sizeof(M##_T))`
- `#define MESSAGE_DATA_BULID(M, S) M##_T *msg = PanicUnlessMalloc(sizeof(M##_T) + S)`
- `#define MESSAGE_OFFSET(M) ((UINT8 *)msg + sizeof(M##_T))`
- `#define T_HOUR(h) ((UINT32)((h) * (UINT32)1000 * (UINT32)60) * (UINT32)60)`
- `#define T_MIN(m) ((UINT32)((m) * (UINT32)1000 * (UINT32)60))`
- `#define T_SEC(s) ((UINT32)((s) * (UINT32)1000))`

Typedefs

- `typedef MsgData MESSAGE`
- `typedef UINT16 MESSAGEID`
- `typedef void const * MsgData`
- `typedef const UINT8 * MsgLock`
- `typedef MsgLock MSGLOCK`
- `typedef UINT16 MSGSUBID`
- `typedef UINT32 MSGTIMER`
- `typedef TASKPACK * Task`
- `typedef Task TASK`
- `typedef void(* TASKHANDLER) (Task, UINT16, MsgData)`
- `typedef void ** TASKPACK`

Enumerations

- `enum { LE_SYS_MSG_BUF_OVERFLOW = (LE_SYS_MSG_BASE + 1), LE_SYS_MSG_TOP }`
BLE system message id.

Functions

- UINT16 [LeCancelAllMessage](#) (TASK task, MESSAGEID id)
Cancel all message in queue.
- UINT16 [LeCancelAllSubMessage](#) (TASK task, MESSAGEID id, MSGSUBID subId)
Cancel all sub message in queue.
- BOOL [LeCancelFirstMessage](#) (TASK task, MESSAGEID id)
Cancel the first message in queue.
- BOOL [LeCancelFirstSubMessage](#) (TASK task, MESSAGEID id, MSGSUBID subId)
Cancel the first sub message in queue.
- UINT16 [LeGetSubMsgId](#) (UINT16 *s)
Get sub message id.
- BOOL [LeHostCreateTask](#) (TASK task, TASKHANDLER hdl)
Create BLE task.
- void [LeHostMessageLoop](#) (void)
message loop run.
- void [LeSendMessage](#) (TASK task, MESSAGEID msgId, MESSAGE msg)
Send message to BLE task.
- void [LeSendMessageAfter](#) (TASK task, MESSAGEID msgId, MESSAGE msg, UINT32 delay)
Delay, then send message to BLE task.
- void [LeSendMessageUnlock](#) (TASK task, MESSAGEID id, MESSAGE msg, MSGLOCK lock)
Send message until lock is 0.
- void [LeSendSubMessage](#) (TASK task, MESSAGEID msgId, MSGSUBID subId, MESSAGE msg)
Send sub message.
- void [LeSendSubMessageAfter](#) (TASK task, MESSAGEID msgId, MSGSUBID subId, MESSAGE msg, UINT32 delay)
Delay, then send sub message.
- void [LeSendSubMessageUnlock](#) (TASK task, MESSAGEID id, MSGSUBID subId, MESSAGE msg, MSGLOCK lock)
Send sub message until lock is 0.

4.5.1 Detailed Description

4.5.2 Macro Definition Documentation

4.5.2.1 LE_ATT_MSG_BASE

```
#define LE_ATT_MSG_BASE 0x1400
```

4.5.2.2 LE_CM_MSG_BASE

```
#define LE_CM_MSG_BASE 0x1100
```

4.5.2.3 LE_GATT_MSG_BASE

```
#define LE_GATT_MSG_BASE 0x1500
```

4.5.2.4 LE_HCI_MSG_BASE

```
#define LE_HCI_MSG_BASE 0x1000
```

4.5.2.5 LE_L2CAP_MSG_BASE

```
#define LE_L2CAP_MSG_BASE 0x1200
```

4.5.2.6 LE_SMP_MSG_BASE

```
#define LE_SMP_MSG_BASE 0x1300
```

4.5.2.7 LE_SYS_MSG_BASE

```
#define LE_SYS_MSG_BASE 0x8000
```

4.5.2.8 MESSAGE_ALLOCATE

```
#define MESSAGE_ALLOCATE(  
    M,  
    S ) PanicUnlessMalloc(sizeof(M##_T) + S)
```

4.5.2.9 MESSAGE_BULID

```
#define MESSAGE_BULID(  
    M ) M##_T *msg = PanicUnlessMalloc(sizeof(M##_T))
```

4.5.2.10 MESSAGE_DATA_BULID

```
#define MESSAGE_DATA_BULID(  
    M,  
    S ) M##_T *msg = PanicUnlessMalloc(sizeof(M##_T) + S)
```

4.5.2.11 MESSAGE_OFFSET

```
#define MESSAGE_OFFSET(  
    M ) ((UINT8 *)msg + sizeof(M##_T))
```

4.5.2.12 T_HOUR

```
#define T_HOUR(  
    h ) ((UINT32)(h) * (UINT32)1000 * (UINT32)60 * (UINT32)60)
```

4.5.2.13 T_MIN

```
#define T_MIN(  
    m ) ((UINT32)(m) * (UINT32)1000 * (UINT32)60)
```

4.5.2.14 T_SEC

```
#define T_SEC(  
    s ) ((UINT32)(s) * (UINT32)1000)
```

4.5.3 Typedef Documentation

4.5.3.1 MESSAGE

```
typedef MsgData MESSAGE
```


4.5.3.2 MESSAGEID

```
typedef UINT16 MESSAGEID
```

4.5.3.3 MsgData

```
typedef void const* MsgData
```

4.5.3.4 MsgLock

```
typedef const UINT8* MsgLock
```

4.5.3.5 MSGLOCK

```
typedef MsgLock MSGLOCK
```

4.5.3.6 MSGSUBID

```
typedef UINT16 MSGSUBID
```

4.5.3.7 MSGTIMER

```
typedef UINT32 MSGTIMER
```

4.5.3.8 Task

```
typedef TASKPACK* Task
```

4.5.3.9 TASK

```
typedef Task TASK
```

4.5.3.10 TASKHANDLER

```
typedef void(* TASKHANDLER) (Task, UINT16, MsgData)
```

4.5.3.11 TASKPACK

```
typedef void** TASKPACK
```

4.5.4 Enumeration Type Documentation

4.5.4.1 anonymous enum

anonymous enum

BLE system message id.

Enumerator

LE_SYS_MSG_BUF_OVERFLOW	message buffer overflow
LE_SYS_MSG_TOP	top of system message id

4.5.5 Function Documentation

4.5.5.1 LeCancelAllMessage()

```
UINT16 LeCancelAllMessage (
    TASK task,
    MESSAGEID id )
```

Cancel all message in queue.

Parameters

<i>task</i>	task.
<i>id</i>	message id.

Returns

0 is ok, others is error.

4.5.5.2 LeCancelAllSubMessage()

```
UINT16 LeCancelAllSubMessage (
    TASK task,
    MESSAGEID id,
    MSGSUBID subId )
```

Cancel all sub message in queue.

Parameters

<i>task</i>	the task of recvice message.
<i>id</i>	message id.
<i>sub↔ id</i>	sub message id.

Returns

0 is ok, others is error.

4.5.5.3 LeCancelFirstMessage()

```
BOOL LeCancelFirstMessage (
    TASK task,
    MESSAGEID id )
```

Cancel the first message in queue.

Parameters

<i>task</i>	task.
<i>id</i>	message id.

Returns

True is ok, false is error.

4.5.5.4 LeCancelFirstSubMessage()

```
BOOL LeCancelFirstSubMessage (
    TASK task,
    MESSAGEID id,
    MSGSUBID subId )
```

Cancel the first sub message in queue.

Parameters

<i>task</i>	the task of recvice message.
<i>id</i>	message id.
<i>subId</i>	sub message id.

Returns

True is ok, false is error.

4.5.5.5 LeGetSubMsgId()

```
UINT16 LeGetSubMsgId (
    UINT16 * s )
```

Get sub message id.

Parameters

<i>sub</i>	message id.
------------	-------------

Returns

0 is ok, others is error.

4.5.5.6 LeHostCreateTask()

```
BOOL LeHostCreateTask (
    TASK task,
    TASKHANDLER hdl )
```

Create BLE task.

Parameters

<i>task</i>	the reference of BLE task.
<i>hdl</i>	callback handle of BLE task.

Returns

TRUE is success, FALSE is failed.

4.5.5.7 LeHostMessageLoop()

```
void LeHostMessageLoop (
    void )
```

message loop run.

Returns

None.

4.5.5.8 LeSendMessage()

```
void LeSendMessage (
    TASK task,
    MESSAGEID msgId,
    MESSAGE msg )
```

Send message to BLE task.

Parameters

<i>task</i>	reference of BLE task.
<i>msgId</i>	message ID.
<i>msg</i>	message.

Returns

None.

4.5.5.9 LeSendMessageAfter()

```
void LeSendMessageAfter (
    TASK task,
```

```

MESSAGEID msgId,
MESSAGE msg,
UINT32 delay )

```

Delay, then send message to BLE task.

Parameters

<i>task</i>	reference of BLE task.
<i>msg</i> ↔ <i>Id</i>	message ID.
<i>msg</i>	message.
<i>delay</i>	delay time, ms.

Returns

None.

4.5.5.10 LeSendMessageUnlock()

```

void LeSendMessageUnlock (
    TASK task,
    MESSAGEID id,
    MESSAGE msg,
    MSGLOCK lock )

```

Send message until lock is 0.

Parameters

<i>task</i>	the task of recvice message.
<i>id</i>	message id.
<i>msg</i>	message.
<i>lock</i>	lock number.

Returns

None.

4.5.5.11 LeSendSubMessage()

```

void LeSendSubMessage (
    TASK task,
    MESSAGEID msgId,
    MSGSUBID subId,
    MESSAGE msg )

```

Send sub message.

Parameters

<i>task</i>	the task of recvice message.
<i>msg↔ Id</i>	message id.
<i>subId</i>	sub message id.
<i>msg</i>	message.

Returns

None.

4.5.5.12 LeSendSubMessageAfter()

```
void LeSendSubMessageAfter (
    TASK task,
    MESSAGEID msgId,
    MSGSUBID subId,
    MESSAGE msg,
    UINT32 delay )
```

Delay, then send sub message.

Parameters

<i>task</i>	the task of recvice message.
<i>msg↔ Id</i>	message id.
<i>subId</i>	sub message id.
<i>msg</i>	message.
<i>delay</i>	delay time.

Returns

None.

4.5.5.13 LeSendSubMessageUnlock()

```
void LeSendSubMessageUnlock (
    TASK task,
    MESSAGEID id,
    MSGSUBID subId,
    MESSAGE msg,
    MSGLOCK lock )
```

Send sub message until lock is 0.

Parameters

<i>task</i>	the task of recvice message.
<i>id</i>	message id.
<i>sub↔ id</i>	sub message id.
<i>msg</i>	message.
<i>lock</i>	lock number.

Returns

None.

4.6 BLE SMP APIs

Data Structures

- struct [LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T](#)
- struct [LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T](#)
- struct [LE_SMP_MSG_OOB_DATA_REQUEST_IND_T](#)
- struct [LE_SMP_MSG_PAIRING_ACTION_IND_T](#)
- struct [LE_SMP_MSG_PAIRING_COMPLETE_IND_T](#)
- struct [LE_SMP_MSG_PASSKEY_DISPLAY_IND_T](#)
- struct [LE_SMP_MSG_PASSKEY_INPUT_IND_T](#)
- struct [LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND_T](#)
- struct [LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T](#)
- struct [LE_SMP_MSG_USER_CONFIRM_IND_T](#)
- struct [LE_SMP_SC_OOB_DATA_T](#)

Macros

- `#define LE_MAX_BOND_COUNT 8`
- `#define LE_SM_IO_CAP_DISP_ONLY 0x00`
- `#define LE_SM_IO_CAP_DISP_YES_NO 0x01`
- `#define LE_SM_IO_CAP_KEYBOARD_DISP 0x04`
- `#define LE_SM_IO_CAP_KEYBOARD_ONLY 0x02`
- `#define LE_SM_IO_CAP_NO_IO 0x03`
- `#define LE_SM_PAIR_MITM_NO 0x00`
- `#define LE_SM_PAIR_MITM_YES 0x01`
- `#define LE_SM_PAIR_OOB_NO 0x00`
- `#define LE_SM_PAIR_OOB_YES 0x01`
- `#define LE_SM_PAIR_SC_NO 0x00`
- `#define LE_SM_PAIR_SC_YES 0x01`

Enumerations

- enum {
[LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND](#) = [LE_SMP_MSG_BASE](#),
[LE_SMP_MSG_PAIRING_ACTION_IND](#),
[LE_SMP_MSG_PASSKEY_DISPLAY_IND](#), [LE_SMP_MSG_PASSKEY_INPUT_IND](#),
[LE_SMP_MSG_OOB_DATA_REQUEST_IND](#), [LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND](#),
[LE_SMP_MSG_USER_CONFIRM_IND](#) [LE_SMP_MSG_ENCRYPTION_CHANGE_IND](#),
[LE_SMP_MSG_ENCRYPTION_REFRESH_IND](#), [LE_SMP_MSG_PAIRING_COMPLETE_IND](#),
[LE_SMP_LONG_TERM_KEY_REQ](#),
[LE_SMP_KEYS_IND](#),
[LE_SMP_MSG_TOP](#) }
BLE SMP message id.
- enum {
[LE_SMP_PAIR_JUST_WORK](#), [LE_SMP_PAIR_OOB](#), [LE_SMP_PAIR_PASSKEY_INPUT](#), [LE_SMP_PAIR_DISPLAY](#),
[LE_SMP_PAIR_NUM_COMPARE](#) }

Functions

- void [LeSmpInit](#) (TASK appTask)
BLE SMP Module Init.
- void [LeSmpOobAuthDataRsp](#) (UINT16 conn_hdl, UINT8 *data, UINT16 len)
SMP OOB authenticate data response.
- UINT16 [LeSmpOobPresent](#) (UINT16 conn_hdl, BOOL oob_present)
SMP OOB present.
- void [LeSmpPasskeyInput](#) (UINT16 conn_hdl, UINT32 passkey)
Input passkey.
- UINT16 [LeSmpScOobComputeConfirmVal](#) (UINT8 *rand, UINT8 *confirm)
SMP secure connection OOB compute confirm value.
- void [LeSmpScOobDataRsp](#) (UINT16 conn_hdl, UINT8 *our_rand, [LE_SMP_SC_OOB_DATA_T](#) *peer)
OOB data response.
- UINT16 [LeSmpSecurityReq](#) (UINT16 conn_hdl)
BLE SMP security request.
- UINT16 [LeSmpSecurityRsp](#) (UINT16 conn_hdl, BOOL accept)
BLE SMP security request.
- UINT16 [LeSmpSetDefaultConfig](#) (UINT8 iocap, BOOL mitm, BOOL sc, BOOL bond)
Set default configure for pairing.
- UINT16 [LeSmpUserConfirmRsp](#) (UINT16 conn_hdl, BOOL accept)
User confirm response.

4.6.1 Detailed Description

4.6.2 Macro Definition Documentation

4.6.2.1 LE_MAX_BOND_COUNT

```
#define LE_MAX_BOND_COUNT 8
```

4.6.2.2 LE_SM_IO_CAP_DISP_ONLY

```
#define LE_SM_IO_CAP_DISP_ONLY 0x00
```

display only

4.6.2.3 LE_SM_IO_CAP_DISP_YES_NO

```
#define LE_SM_IO_CAP_DISP_YES_NO 0x01
```

display + yes or no

4.6.2.4 LE_SM_IO_CAP_KEYBOARD_DISP

```
#define LE_SM_IO_CAP_KEYBOARD_DISP 0x04
```

display + keyboard

4.6.2.5 LE_SM_IO_CAP_KEYBOARD_ONLY

```
#define LE_SM_IO_CAP_KEYBOARD_ONLY 0x02
```

keyboard only

4.6.2.6 LE_SM_IO_CAP_NO_IO

```
#define LE_SM_IO_CAP_NO_IO 0x03
```

no input and output

4.6.2.7 LE_SM_PAIR_MITM_NO

```
#define LE_SM_PAIR_MITM_NO 0x00
```

4.6.2.8 LE_SM_PAIR_MITM_YES

```
#define LE_SM_PAIR_MITM_YES 0x01
```

4.6.2.9 LE_SM_PAIR_OOB_NO

```
#define LE_SM_PAIR_OOB_NO 0x00
```

4.6.2.10 LE_SM_PAIR_OOB_YES

```
#define LE_SM_PAIR_OOB_YES 0x01
```

4.6.2.11 LE_SM_PAIR_SC_NO

```
#define LE_SM_PAIR_SC_NO 0x00
```

4.6.2.12 LE_SM_PAIR_SC_YES

```
#define LE_SM_PAIR_SC_YES 0x01
```

4.6.3 Enumeration Type Documentation

4.6.3.1 anonymous enum

```
anonymous enum
```

BLE SMP message id.

Enumerator

LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND	slave security request
LE_SMP_MSG_PAIRING_ACTION_IND	pairing action indication
LE_SMP_MSG_PASSKEY_DISPLAY_IND	passkey display indication
LE_SMP_MSG_PASSKEY_INPUT_IND	passkey input indication
LE_SMP_MSG_OOB_DATA_REQUEST_IND	OOB data request indication
LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND	SC OOB data request indication
LE_SMP_MSG_USER_CONFIRM_IND	user confirm indication
LE_SMP_MSG_ENCRYPTION_CHANGE_IND	encryption change indication
LE_SMP_MSG_ENCRYPTION_REFRESH_IND	encryption refresh indication
LE_SMP_MSG_PAIRING_COMPLETE_IND	pairing complete indication
LE_SMP_LONG_TERM_KEY_REQ	long term key request
LE_SMP_KEYS_IND	keys indication
LE_SMP_MSG_TOP	top of SMP message id

4.6.3.2 anonymous enum

```
anonymous enum
```

Enumerator

LE_SMP_PAIR_JUST_WORK	just work
LE_SMP_PAIR_OOB	out of band
LE_SMP_PAIR_PASSKEY_INPUT	passkey entry
LE_SMP_PAIR_DISPLAY	display
LE_SMP_PAIR_NUM_COMPARE	number compare

4.6.4 Function Documentation

4.6.4.1 LeSmpInit()

```
void LeSmpInit (
    TASK appTask )
```

BLE SMP Module Init.

Parameters

<i>appTask</i>	the reference of BLE task.
----------------	----------------------------

Returns

None.

4.6.4.2 LeSmpOobAuthDataRsp()

```
void LeSmpOobAuthDataRsp (
    UINT16 conn_hdl,
    UINT8 * data,
    UINT16 len )
```

SMP OOB authenticate data response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>data</i>	response data.
<i>len</i>	data length.

Returns

None.

4.6.4.3 LeSmpOobPresent()

```
UINT16 LeSmpOobPresent (
    UINT16 conn_hdl,
    BOOL oob_present )
```

SMP OOB present.

Parameters

<i>conn_hdl</i>	connection handle.
<i>oob_present</i>	present or not.

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.6.4.4 LeSmpPasskeyInput()

```
void LeSmpPasskeyInput (
    UINT16 conn_hdl,
    UINT32 passkey )
```

Input passkey.

Parameters

<i>conn_hdl</i>	connection handle.
<i>passkey</i>	passkey.

Returns

None.

4.6.4.5 LeSmpScOobComputeConfirmVal()

```
UINT16 LeSmpScOobComputeConfirmVal (
    UINT8 * rand,
    UINT8 * confirm )
```

SMP secure connection OOB compute confirm value.

Parameters

<i>rand</i>	random data.
<i>confirm</i>	confirm data.

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.6.4.6 LeSmpScOobDataRsp()

```
void LeSmpScOobDataRsp (
    UINT16 conn_hdl,
    UINT8 * our_rand,
    LE_SMP_SC_OOB_DATA_T * peer )
```

OOB data response.

Parameters

<i>conn_hdl</i>	connection handld.
<i>our_rand</i>	our random data.
<i>peer</i>	peer OOB data.

Returns

None.

4.6.4.7 LeSmpSecurityReq()

```
UINT16 LeSmpSecurityReq (
    UINT16 conn_hdl )
```

BLE SMP security request.

Parameters

<i>conn_hdl</i>	connection handle.
-----------------	--------------------

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.6.4.8 LeSmpSecurityRsp()

```
UINT16 LeSmpSecurityRsp (
    UINT16 conn_hdl,
    BOOL accept )
```

BLE SMP security request.

Parameters

<i>conn_hdl</i>	connection handle.
<i>accept</i>	TRUE is accept, FALSE is not.

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.6.4.9 LeSmpSetDefaultConfig()

```
UINT16 LeSmpSetDefaultConfig (
    UINT8 iocap,
    BOOL mitm,
    BOOL sc,
    BOOL bond )
```

Set default configure for pairing.

Parameters

<i>iocap</i>	IO capability.
<i>mitm</i>	TRUE is MITM protected, FALSE is not.
<i>sc</i>	TRUE is request BLE secure connection pairing, FALSE is not.
<i>bond</i>	TRUE: bonding, FALSE: no bonding.

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.6.4.10 LeSmpUserConfirmRsp()

```
UINT16 LeSmpUserConfirmRsp (
    UINT16 conn_hdl,
    BOOL accept )
```

User confirm response.

Parameters

<i>conn_hdl</i>	connection handle.
<i>accept</i>	yes or no.

Returns

0 is Ok, others refer to SMP_ERR_* in [ble_err.h](#).

4.7 WIFI APIS

WIFI APIS.

Modules

- [WIFI Common APIs](#)
- [WIFI STA APIs](#)
- [Enumeration](#)

Data Structures

- struct [wifi_active_scan_time_t](#)
Range of active scan times per channel.
- struct [wifi_ap_config_t](#)
This structure is the Wi-Fi configuration for initialization for Soft-AP mode.
- struct [wifi_auto_connect_info_t](#)
WiFi auto connect info parameters.
- union [wifi_config_t](#)
Wi-Fi configuration for initialization.
- struct [wifi_fast_scan_threshold_t](#)
Structure describing parameters for a Wi-Fi fast scan.
- struct [wifi_init_config_t](#)
WiFi stack configuration parameters.
- struct [wifi_scan_config_t](#)
Parameters for an SSID scan.
- struct [wifi_scan_info_t](#)
This structure defines the inforamtion of scanned APs.
- struct [wifi_scan_list_t](#)
This structure defines the list of scanned APs with their corresponding information.
- union [wifi_scan_time_t](#)
Aggregate of active & passive scan time per channel.
- struct [wifi_sta_config_t](#)
This structure is the Wi-Fi configuration for initialization for STA mode.

Macros

- #define [WIFI_BEACON_INTERVAL_LENGTH](#) (2)
Beacon interval length in a frame header.
- #define [WIFI_CAPABILITY_INFO_LENGTH](#) (2)
Length of capability information in a frame header.
- #define [WIFI_LENGTH_802_11](#) (24)
Length of 802.11 MAC header.
- #define [WIFI_LENGTH_PASSPHRASE](#) (64)
The maximum length of passphrase used in WPA-PSK and WPA2-PSK encryption types.
- #define [WIFI_MAC_ADDRESS_LENGTH](#) (6)
MAC address length.
- #define [WIFI_MAX_LENGTH_OF_SSID](#) (32+1)
The maximum length of SSID.
- #define [WIFI_MAX_SCAN_AP_NUM](#) (16)
maximum number of ap list items which can stored
- #define [WIFI_MAX_SUPPORTED_RATES](#) (8)
maximum number of supported rates which can used

Typedefs

- typedef int(* [wifi_event_notify_cb_t](#)) (void *data)

Functions

- int [wifi_event_process_handler](#) ([wifi_event_t](#) event, uint8_t *payload, uint32_t length)
Default event handler for system events.
- void [wifi_install_default_event_handlers](#) (void)
Set discoverability and connectability mode for legacy bluetooth. This function should.
- int [wifi_register_event_handler](#) ([wifi_event_t](#) idx, [wifi_event_handler_t](#) handler)
Set discoverability and connectability mode for legacy bluetooth. This function should.

4.7.1 Detailed Description

WIFI APIs.

4.7.2 Macro Definition Documentation

4.7.2.1 WIFI_BEACON_INTERVAL_LENGTH

```
#define WIFI_BEACON_INTERVAL_LENGTH (2)
```

Beacon interval length in a frame header.

4.7.2.2 WIFI_CAPABILITY_INFO_LENGTH

```
#define WIFI_CAPABILITY_INFO_LENGTH (2)
```

Length of capability information in a frame header.

4.7.2.3 WIFI_LENGTH_802_11

```
#define WIFI_LENGTH_802_11 (24)
```

Length of 802.11 MAC header.

4.7.2.4 WIFI_LENGTH_PASSPHRASE

```
#define WIFI_LENGTH_PASSPHRASE (64)
```

The maximum length of passphrase used in WPA-PSK and WPA2-PSK encryption types.

4.7.2.5 WIFI_MAC_ADDRESS_LENGTH

```
#define WIFI_MAC_ADDRESS_LENGTH (6)
```

MAC address length.

4.7.2.6 WIFI_MAX_LENGTH_OF_SSID

```
#define WIFI_MAX_LENGTH_OF_SSID (32+1)
```

The maximum length of SSID.

4.7.2.7 WIFI_MAX_SCAN_AP_NUM

```
#define WIFI_MAX_SCAN_AP_NUM (16)
```

maximum number of ap list items which can stored

4.7.2.8 WIFI_MAX_SUPPORTED_RATES

```
#define WIFI_MAX_SUPPORTED_RATES (8)
```

maximum number of supported rates which can used

4.7.3 Typedef Documentation

4.7.3.1 wifi_event_notify_cb_t

```
typedef int (* wifi_event_notify_cb_t) (void *data)
```

4.7.4 Function Documentation

4.7.4.1 wifi_event_process_handler()

```
int wifi_event_process_handler (
    wifi_event_t event,
    uint8_t * payload,
    uint32_t length )
```

Default event handler for system events.

This function performs default handling of system events. When using event_loop APIs, it is called automatically before invoking the user-provided callback function.

Applications which implement a custom event loop must call this function as part of event processing.

Parameters

in	<i>event</i>	event type Set the event type,Options are <ul style="list-style-type: none"> • WIFI_EVENT_INIT_COMPLETE • WIFI_EVENT_SCAN_COMPLETE • WIFI_EVENT_STA_START • WIFI_EVENT_STA_STOP • WIFI_EVENT_STA_CONNECTED • WIFI_EVENT_STA_DISCONNECTED • WIFI_EVENT_STA_CONNECTION_FAILED • WIFI_EVENT_STA_GOT_IP
in	<i>payload</i>	Data block that transmitted to event
in	<i>length</i>	The length of data block

Returns

0 : success
other : failed

4.7.4.2 wifi_install_default_event_handlers()

```
void wifi_install_default_event_handlers (
    void )
```

Set discoverability and connectability mode for legacy bluetooth. This function should.

4.7.4.3 wifi_register_event_handler()

```
int wifi_register_event_handler (
    wifi_event_t idx,
    wifi_event_handler_t handler )
```

Set discoverability and connectability mode for legacy bluetooth. This function should.

Parameters

in	<i>idx</i>	one of the enums of bt_scan_mode_t
in	<i>handler</i>	the Wi-Fi event handler

Returns

0 : success
other : failed

4.8 WIFI Common APIs

Data Structures

- struct `event_msg_t`
Send information to event by `event_msg_t`.
- union `wifi_event_info_t`
`wifi_event_info_t`
- struct `wifi_event_sta_connected_t`
`wifi_event_sta_connected_t`
- struct `wifi_event_sta_disconnected_t`
`wifi_event_sta_disconnected_t`
- struct `wifi_event_sta_got_ip_t`
`wifi_event_sta_got_ip_t`
- struct `wifi_event_sta_scan_done_t`
`wifi_event_sta_scan_done_t`

Typedefs

- typedef int(* `wifi_event_cb_t`) (`wifi_event_id_t` event, void *data, uint16_t length)
Application specified event callback function.

Functions

- int `wifi_event_loop_init` (`wifi_event_cb_t` cb)
Event Loop Initialization Create the event handler and call back funtion.
- int `wifi_event_loop_send` (`event_msg_t` *msg)
Send an event to event task.
- void `wifi_event_loop_set_cb` (`wifi_event_cb_t` cb, void *ctx)
Set application specified event callback function.
- int `wifi_event_process_handler` (`wifi_event_t` event, uint8_t *payload, uint32_t length)
Default event handler for system events.

4.8.1 Detailed Description

4.8.2 Typedef Documentation

4.8.2.1 `wifi_event_cb_t`

```
typedef int(* wifi_event_cb_t) (wifi_event_id_t event, void *data, uint16_t length)
```

Application specified event callback function.

4.8.3 Function Documentation

4.8.3.1 wifi_event_loop_init()

```
int wifi_event_loop_init (  
    wifi_event_cb_t cb )
```

Event Loop Initialization Create the event handler and call back funtion.

Parameters

<i>cb</i>	: application specified event callback
-----------	--

Returns

0 : success
other : failed

4.8.3.2 wifi_event_loop_send()

```
int wifi_event_loop_send (
    event_msg_t * msg )
```

Send an event to event task.

Attention

1. Other task/modules, such as the TCP/IP module, can call this API to send an event to event task

Parameters

<i>event_msg_t</i>	* msg: Send information to event by msg
--------------------	---

Returns

0 : success
other : failed

4.8.3.3 wifi_event_loop_set_cb()

```
void wifi_event_loop_set_cb (
    wifi_event_cb_t cb,
    void * ctx )
```

Set application specified event callback function.

Attention

1. If cb is NULL, means application does not need to handle. If cb is not NULL, it will be called when an event is received and after the default event callback is completed

Parameters

<i>wifi_event_cb_t</i>	cb : callback
<i>void</i>	*ctx : reserved for user

4.8.3.4 wifi_event_process_handler()

```
int wifi_event_process_handler (
    wifi_event_t event,
    uint8_t * payload,
    uint32_t length )
```

Default event handler for system events.

This function performs default handling of system events.

Applications which implement a custom event loop must call this function as part of event processing.

Parameters

in	<i>event</i>	event type Set the event type,Options are <ul style="list-style-type: none"> • WIFI_EVENT_INIT_COMPLETE • WIFI_EVENT_SCAN_COMPLETE • WIFI_EVENT_STA_START • WIFI_EVENT_STA_STOP • WIFI_EVENT_STA_CONNECTED • WIFI_EVENT_STA_DISCONNECTED • WIFI_EVENT_STA_CONNECTION_FAILED • WIFI_EVENT_STA_GOT_IP
in	<i>payload</i>	Data block transmitted to event
in	<i>length</i>	The length of the data block

Returns

0 : success
other : failed

4.9 WIFI STA APIs

Typedefs

- typedef int32_t(* [wifi_event_handler_t](#)) ([wifi_event_t](#) event, uint8_t *payload, uint32_t length)
This defines the Wi-Fi event handler. Call [wifi_connection_register_event_handler\(\)](#) to register a handler, then the Wi-Fi driver generates an event and sends it to the handler.
- typedef void(* [wifi_init_complete_cb_t](#)) (void *ctx)
Initialization of complete callback function.
- typedef int32_t [wifi_result_t](#)

Functions

- int [wifi_auto_connect_del_ap_info](#) (u8 index)
Delete automatically connected AP information stored in flash.
- int [wifi_auto_connect_get_ap_info](#) (u8 index, [wifi_auto_connect_info_t](#) *info)
Get ap detailed information saved in flash.
- u8 [wifi_auto_connect_get_ap_num](#) (void)
Get the number of automatically connected aps that have been saved in the flash.
- u8 [wifi_auto_connect_get_mode](#) (void)
Get the status of the current automatic connection mode.
- int [wifi_auto_connect_init](#) (void)
Initialize wifi automatic connection.
- int [wifi_auto_connect_reset](#) (void)
Rest the auto connect process.
- int [wifi_auto_connect_set_ap_num](#) (u8 num)
Save the number of automatically connected ap to flash.
- int [wifi_auto_connect_set_mode](#) (u8 mode)
Set the connection type.
- int [wifi_auto_connect_start](#) (void)
Start wifi automatic connection process.
- int [wifi_config_get_bandwidth](#) ([wifi_mode_t](#) interface, [wifi_bandwidth_t](#) *bandwidth)
Get the bandwidth of OPL1000 specified interface.
- int [wifi_config_get_bssid](#) (uint8_t *bssid)
get bssid after scan
- int [wifi_config_get_channel](#) ([wifi_mode_t](#) interface, uint8_t *channel)
Get the primary/secondary channel of OPL1000.
- int [wifi_config_get_dtim_interval](#) (uint8_t *interval)
Get the interval of DTIM.
- int [wifi_config_get_listen_interval](#) (uint8_t *interval)
Get the interval of listen.
- int [wifi_config_get_mac_address](#) ([wifi_mode_t](#) interface, uint8_t *address)
Get mac of specified interface.
- int [wifi_config_get_opmode](#) (uint8_t *mode)
Set wifi operation mode.
- int [wifi_config_get_skip_dtim](#) (uint8_t *value)
Get the Skip DTIM value in current wifi setting of OPL1000.
- int [wifi_config_get_ssid](#) (uint8_t *ssid, uint8_t *ssid_length)
Get ssid value of AP.
- int [wifi_config_set_bandwidth](#) ([wifi_mode_t](#) interface, [wifi_bandwidth_t](#) bandwidth)

- Set the bandwidth of OPL1000 specified interface.*

 - int [wifi_config_set_bssid](#) (uint8_t *bssid)
config OPL1000 Wi-Fi bssid.
 - int [wifi_config_set_channel](#) ([wifi_mode_t](#) interface, uint8_t channel)
Set primary/secondary channel of OPL1000.
 - int [wifi_config_set_dtim_interval](#) (uint8_t interval)
Set the interval of DTIM.
 - int [wifi_config_set_listen_interval](#) (uint8_t interval)
Set the interval of listen.
 - int [wifi_config_set_mac_address](#) ([wifi_mode_t](#) interface, uint8_t *address)
Set MAC address of OPL1000 Wi-Fi station or the soft-AP interface.
 - int [wifi_config_set_opmode](#) (uint8_t mode)
Set wifi operation mode.
 - int [wifi_config_set_skip_dtim](#) (uint8_t value)
Set the Skip DTIM value of OPL1000.
 - int [wifi_config_set_ssid](#) ([wifi_mode_t](#) interface, uint8_t *ssid, uint8_t ssid_length)
Set the ssid value of the current device.
 - int [wifi_connection_connect](#) ([wifi_config_t](#) *config)
Connect OPL1000 Wi-Fi station to certain AP.
 - int [wifi_connection_disconnect_ap](#) (void)
Disconnect the link between OPL1000 and connected AP.
 - int [wifi_connection_disconnect_sta](#) (uint8_t *address)
Disconnect the link between the current device and the station.
 - int [wifi_connection_get_rssi](#) (int8_t *rssi)
get signal strength of AP
 - int [wifi_connection_register_event_handler](#) ([wifi_event_t](#) event, [wifi_event_handler_t](#) handler)
register wifi call back handler
 - int [wifi_connection_scan_start](#) (uint8_t *ssid, uint8_t ssid_length, uint8_t *bssid, uint8_t scan_mode, uint8_t scan_option)
Scan start.
 - int [wifi_connection_unregister_event_handler](#) ([wifi_event_t](#) event, [wifi_event_handler_t](#) handler)
unregister wifi call back handler
 - int [wifi_deinit](#) (void)
De-init Wi-Fi Initialization and Configuration functions.
 - u8 [wifi_fast_connect_get_mode](#) (u8 ap_index)
Get the status of AP fast connection.
 - int [wifi_fast_connect_set_mode](#) (u8 mode, u8 ap_index)
Set the fast connection type.
 - int [wifi_fast_connect_start](#) (void)
Start the fast connection process.
 - int [wifi_get_config](#) ([wifi_mode_t](#) interface, [wifi_config_t](#) *conf)
Get configuration of specified interface.
 - int [wifi_init](#) (const [wifi_init_config_t](#) *config, [wifi_init_complete_cb_t](#) init_cb)
Init Wi-Fi Initializes the wifi according to the specified parameters in the config.
 - int [wifi_scan_get_ap_list](#) ([wifi_scan_list_t](#) *scan_list)
Get list of APs that found in last scan operation.
 - int [wifi_scan_get_ap_num](#) (uint16_t *number)
Get the number of scanned APs.
 - int [wifi_scan_get_ap_records](#) (uint16_t *number, [wifi_scan_info_t](#) *ap_records)
Get AP list found in last scan operation.
 - int [wifi_scan_scan_stop](#) (void)

- Stop scanning process.*

 - int [wifi_scan_start](#) (const [wifi_scan_config_t](#) *config, bool block)

Scan all available APs. After invoke the [wifi_set_config\(\)](#) and [wifi_start\(\)](#), then call [wifi_scan_start\(\)](#) to scan APs.
- int [wifi_set_config](#) ([wifi_mode_t](#) interface, [wifi_config_t](#) *conf)

Set configuration of OPL1000 STA.
- int [wifi_sta_get_ap_info](#) ([wifi_scan_info_t](#) *ap_info)

Get information of AP which OPL1000 station is associated with.
- int [wifi_start](#) (void)

Start Wi-Fi working.
- int [wifi_stop](#) (void)

Stop wifi working.

4.9.1 Detailed Description

4.9.2 Typedef Documentation

4.9.2.1 [wifi_event_handler_t](#)

```
typedef int32_t(* wifi_event_handler_t) (wifi\_event\_t event, uint8_t *payload, uint32_t length)
```

This defines the Wi-Fi event handler. Call [wifi_connection_register_event_handler\(\)](#) to register a handler, then the Wi-Fi driver generates an event and sends it to the handler.

Parameters

in	<i>event</i>	is an optional event to register. For more details, please refer to wifi_event_t .
in	<i>payload</i>	is the payload for the event. When the event is WIFI_EVENT_IOT_CONNECTED in AP mode, payload is the connected STA's MAC address. When the event is WIFI_EVENT_IOT_CONNECTED in STA mode, payload is the connected AP's BSSID.
in	<i>length</i>	is the length of a packet.

Returns

The return value is reserved and it is ignored.

4.9.2.2 [wifi_init_complete_cb_t](#)

```
typedef void(* wifi_init_complete_cb_t) (void *ctx)
```

Initialization of complete callback function.

Invoked when Wi-Fi initialization is complete.

Parameters

<code>ctx</code>	is context pointer that provided to wifi_init() . It will be passed back to the callback.
------------------	---

4.9.2.3 wifi_result_t

```
typedef int32_t wifi_result_t
```

4.9.3 Function Documentation**4.9.3.1 wifi_auto_connect_del_ap_info()**

```
int wifi_auto_connect_del_ap_info (  
    u8 index )
```

Delete automatically connected AP information stored in flash.

Parameters

in	<i>index</i>	: Index of ap information, The range is 0 to 3
----	--------------	--

Returns

0 : success
other : failed

4.9.3.2 wifi_auto_connect_get_ap_info()

```
int wifi_auto_connect_get_ap_info (  
    u8 index,  
    wifi_auto_connect_info_t * info )
```

Get ap detailed information saved in flash.

Parameters

in	<i>index</i>	: Index of ap information, The range is 0 to 3
in	<i>info</i>	: wifi_auto_connect_info_f array to hold the found APs

Returns

0 : success
other : failed

4.9.3.3 wifi_auto_connect_get_ap_num()

```
u8 wifi_auto_connect_get_ap_num (  
    void )
```

Get the number of automatically connected aps that have been saved in the flash.

Returns

0-3 ap number

4.9.3.4 wifi_auto_connect_get_mode()

```
u8 wifi_auto_connect_get_mode (  
    void )
```

Get the status of the current automatic connection mode.

Returns

0 : off
1 : on

4.9.3.5 wifi_auto_connect_init()

```
int wifi_auto_connect_init (  
    void )
```

Initialize wifi automatic connection.

Returns

0 : success
other : failed

4.9.3.6 wifi_auto_connect_reset()

```
int wifi_auto_connect_reset (
    void )
```

Rest the auto connect process.

Returns

0 : success
other : failed

4.9.3.7 wifi_auto_connect_set_ap_num()

```
int wifi_auto_connect_set_ap_num (
    u8 num )
```

Save the number of automatically connected ap to flash.

Parameters

in	Connection	Type
----	------------	------

Returns

0 : success
other : failed

4.9.3.8 wifi_auto_connect_set_mode()

```
int wifi_auto_connect_set_mode (
    u8 mode )
```

Set the connection type.

Parameters

in	Connection	Type
		<ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)

Returns

0 : success
other : failed

4.9.3.9 wifi_auto_connect_start()

```
int wifi_auto_connect_start (
    void )
```

Start wifi automatic connection process.

Returns

0 : success
other : failed

4.9.3.10 wifi_config_get_bandwidth()

```
int wifi_config_get_bandwidth (
    wifi_mode_t interface,
    wifi_bandwidth_t * bandwidth )
```

Get the bandwidth of OPL1000 specified interface.

Attention

1. API returns false if try to get an interface which is not enable

Parameters

in	<i>interface</i>	Configure the current wifi working mode,The options are <ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)
out	<i>bandwidth</i>	Get the bandwidth value of the current wifi module working through the pointer

Returns

0 : success
other : failed

4.9.3.11 wifi_config_get_bssid()

```
int wifi_config_get_bssid (
    uint8_t * bssid )
```

get bssid after scan

Parameters

out	<i>bssid</i>	the string of bssid
-----	--------------	---------------------

Returns

0 : success
other : failed

4.9.3.12 wifi_config_get_channel()

```
int wifi_config_get_channel (
    wifi_mode_t interface,
    uint8_t * channel )
```

Get the primary/secondary channel of OPL1000.

Attention

1. API returns false if try to get an interface which is not enabled

Parameters

in	<i>interface</i>	Configure the current wifi working mode, The options are <ul style="list-style-type: none"> • WIFI_MODE_STA • WIFI_MODE_AP (currently not support)
out	<i>channel</i>	Get Current module wifi work channel number

Returns

0 : success
other : failed

4.9.3.13 wifi_config_get_dtim_interval()

```
int wifi_config_get_dtim_interval (
    uint8_t * interval )
```

Get the interval of DTIM.

Parameters

in	<i>interval</i>	the interval of DTIM
----	-----------------	----------------------

Returns

0 : success
other : failed

4.9.3.14 wifi_config_get_listen_interval()

```
int wifi_config_get_listen_interval (
    uint8_t * interval )
```

Get the interval of listen.

Parameters

in	<i>interval</i>	the interval of listen
----	-----------------	------------------------

Returns

0 : success
other : failed

4.9.3.15 wifi_config_get_mac_address()

```
int wifi_config_get_mac_address (
    wifi_mode_t interface,
    uint8_t * address )
```

Get mac of specified interface.

Parameters

in	<i>interface</i>	Configure the current wifi working mode, The options are <ul style="list-style-type: none"> WIFI_MODE_STA WIFI_MODE_AP (currently not support)
out	<i>address</i>	Get the MAC address of the device through this interface, The address is similar to this structure: xx:xx:xx:xx:xx:xx

Returns

0 : success
other : failed

4.9.3.16 wifi_config_get_opmode()

```
int wifi_config_get_opmode (
    uint8_t * mode )
```

Set wifi operation mode.

Parameters

<i>mode</i>	refer to wifi_mode_t
-------------	--------------------------------------

Returns

0 : success
other : failed

4.9.3.17 wifi_config_get_skip_dtim()

```
int wifi_config_get_skip_dtim (
    uint8_t * value )
```

Get the Skip DTIM value in current wifi setting of OPL1000.

Parameters

out	<i>value</i>	Get the Skip DTIM value in current wifi setting
-----	--------------	---

Returns

0 : success
other : failed

4.9.3.18 wifi_config_get_ssid()

```
int wifi_config_get_ssid (
    uint8_t * ssid,
    uint8_t * ssid_length )
```

Get ssid value of AP.

Parameters

out	<i>ssid</i>	Get ssid by pointer
out	<i>ssid_length</i>	Get the length of the ssid character

Returns

0 : success
other : failed

4.9.3.19 wifi_config_set_bandwidth()

```
int wifi_config_set_bandwidth (
    wifi_mode_t interface,
    wifi_bandwidth_t bandwidth )
```

Set the bandwidth of OPL1000 specified interface.

Parameters

in	<i>interface</i>	Configure the current wifi working mode,The options are <ul style="list-style-type: none"> WIFI_MODE_STA WIFI_MODE_AP (currently not support)
in	<i>bandwidth</i>	Set the working bandwidth of wifi

Returns

0 : success
other : failed

4.9.3.20 wifi_config_set_bssid()

```
int wifi_config_set_bssid (
    uint8_t * bssid )
```

config OPL1000 Wi-Fi bssid.

Parameters

in	<i>bssid</i>	the string of bssid
----	--------------	---------------------

Returns

0 : success
other : failed

4.9.3.21 wifi_config_set_channel()

```
int wifi_config_set_channel (
    wifi_mode_t interface,
    uint8_t channel )
```

Set primary/secondary channel of OPL1000.

Attention

1. This is a special API for sniffer
2. This API should be called after [wifi_start\(\)](#)

Parameters

in	<i>interface</i>	Configure the current wifi working mode,The options are <ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)
in	<i>channel</i>	Set current Wi-Fi work channel number

Returns

0 : success
other : failed

4.9.3.22 wifi_config_set_dtim_interval()

```
int wifi_config_set_dtim_interval (
    uint8_t interval )
```

Set the interval of DTIM.

Parameters

in	<i>interval</i>	the interval of DTIM
----	-----------------	----------------------

Returns

0 : success
other : failed

4.9.3.23 wifi_config_set_listen_interval()

```
int wifi_config_set_listen_interval (
    uint8_t interval )
```

Set the interval of listen.

Parameters

in	<i>interval</i>	the interval of listen
----	-----------------	------------------------

Returns

0 : success
other : failed

4.9.3.24 wifi_config_set_mac_address()

```
int wifi_config_set_mac_address (
    wifi_mode_t interface,
    uint8_t * address )
```

Set MAC address of OPL1000 Wi-Fi station or the soft-AP interface.

Attention

1. This API can only be called when the interface is disabled
2. OPL1000 soft-AP and station have different MAC addresses, do not set them to be the same.

Parameters

in	<i>interface</i>	Configure the current wifi working mode,The options are <ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)
in	<i>address</i>	set MAC address

Returns

0 : success
other : failed

4.9.3.25 wifi_config_set_opmode()

```
int wifi_config_set_opmode (
    uint8_t mode )
```

Set wifi operation mode.

Parameters

<i>mode</i>	refer to wifi_mode_t
-------------	--------------------------------------

Returns

0 : success
other : failed

4.9.3.26 wifi_config_set_skip_dtim()

```
int wifi_config_set_skip_dtim (
    uint8_t value )
```

Set the Skip DTIM value of OPL1000.

Parameters

<i>in</i>	<i>value</i>	Set the Skip DTIM value
-----------	--------------	-------------------------

Attention

1. This API will set the skip DTIM value to share memory and stored in flash, please use [wifi_config_get_skip_dtim\(\)](#) to check it.
2. The setting will be effect after next connect. We recommend re-connect AP after setting to make sure the value is correct.

Returns

0 : success
other : failed

4.9.3.27 wifi_config_set_ssid()

```
int wifi_config_set_ssid (
    wifi_mode_t interface,
    uint8_t * ssid,
    uint8_t ssid_length )
```

Set the ssid value of the current device.

Parameters

in	<i>interface</i>	Configure the current wifi working mode, The options are <ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)
in	<i>ssid</i>	Set the value of ssid
in	<i>ssid_length</i>	The length of ssid parameter

Returns

0 : success
other : failed

4.9.3.28 wifi_connection_connect()

```
int wifi_connection_connect (  
    wifi_config_t * config )
```

Connect OPL1000 Wi-Fi station to certain AP.

Attention

1. This API only impact WIFI_MODE_STA or WIFI_MODE_AP mode
2. If OPL1000 is connected to an AP, call wifi_disconnect to disconnect.

Parameters

in	<i>config</i>	Establish connection parameters
----	---------------	---------------------------------

Returns

0 : success
other : failed

4.9.3.29 wifi_connection_disconnect_ap()

```
int wifi_connection_disconnect_ap (  
    void )
```

Disconnect the link between OPL1000 and connected AP.

Returns

0 : success
other : failed

4.9.3.30 `wifi_connection_disconnect_sta()`

```
int wifi_connection_disconnect_sta (
    uint8_t * address )
```

Disconnect the link between the current device and the station.

Parameters

in	<i>address</i>	station address
----	----------------	-----------------

Returns

0 : success
other : failed

4.9.3.31 `wifi_connection_get_rssi()`

```
int wifi_connection_get_rssi (
    int8_t * rssi )
```

get signal strength of AP

Attention

1. If the scan is successful, this API returns signal strength value, otherwise it will get wrong result

Parameters

out	<i>rssi</i>	rssi value
-----	-------------	------------

Returns

0 : success
other : failed

4.9.3.32 `wifi_connection_register_event_handler()`

```
int wifi_connection_register_event_handler (
    wifi_event_t event,
    wifi_event_handler_t handler )
```

register wifi call back handler

Parameters

in	<i>event</i>	The type of the registered event. Options are <ul style="list-style-type: none"> • WIFI_EVENT_INIT_COMPLETE • WIFI_EVENT_SCAN_COMPLETE • WIFI_EVENT_STA_START • WIFI_EVENT_STA_STOP • WIFI_EVENT_STA_CONNECTED • WIFI_EVENT_STA_DISCONNECTED • WIFI_EVENT_STA_CONNECTION_FAILED • WIFI_EVENT_STA_GOT_IP
in	<i>handler</i>	registered event handler

Returns

0 : success
other : failed

4.9.3.33 wifi_connection_scan_start()

```
int wifi_connection_scan_start (
    uint8_t * ssid,
    uint8_t ssid_length,
    uint8_t * bssid,
    uint8_t scan_mode,
    uint8_t scan_option )
```

Scan start.

Parameters

<i>ssid</i>	ssid string
<i>ssid_length</i>	ssid string length
<i>bssid</i>	bssid
<i>scan_mode</i>	refer to #wifi_scan_mode_ext in wpa_common_patch.h
<i>scan_option</i>	if scan_option is true, this API will block the caller until the scan is done, otherwise it will return immediately

Returns

0 : success
other : failed

4.9.3.34 `wifi_connection_unregister_event_handler()`

```
int wifi_connection_unregister_event_handler (
    wifi_event_t event,
    wifi_event_handler_t handler )
```

unregister wifi call back handler

Parameters

in	<i>event</i>	The type of the unregistered event. Options please refer to wifi_connection_register_event_handler()
in	<i>handler</i>	unregistered event handler

Returns

0 : success
other : failed

4.9.3.35 `wifi_deinit()`

```
int wifi_deinit (
    void )
```

De-init Wi-Fi Initialization and Configuration functions.

Attention

1. This API should be called if want to remove Wi-Fi driver from the system

Returns

0 : success
other : failed

4.9.3.36 `wifi_fast_connect_get_mode()`

```
u8 wifi_fast_connect_get_mode (
    u8 ap_index )
```

Get the status of AP fast connection.

Parameters

in	<i>ap_index</i>	: Index of ap information, The range is 0 to 3
----	-----------------	--

Returns

0 : success
other : failed

4.9.3.37 wifi_fast_connect_set_mode()

```
int wifi_fast_connect_set_mode (
    u8 mode,
    u8 ap_index )
```

Set the fast connection type.

Parameters

in	<i>mode</i>	: Configure the fast connect mode ,0 means disable fast connection, and 1 enable the fast connection mode
in	<i>ap_index</i>	: Index of ap information,The range is 0 to 3

Returns

0 : success
other : failed

4.9.3.38 wifi_fast_connect_start()

```
int wifi_fast_connect_start (
    void )
```

Start the fast connection process.

Returns

0 : success
other : failed

4.9.3.39 wifi_get_config()

```
int wifi_get_config (
    wifi_mode_t interface,
    wifi_config_t * conf )
```

Get configuration of specified interface.

Parameters

in	<i>interface</i>	Configure wifi working mode,The options are <ul style="list-style-type: none"> • WIFI_MODE_STA • WIFI_MODE_AP (currently not support)
out	<i>conf</i>	return wifi's current operating parameters

Returns

0 : success
other : failed

4.9.3.40 wifi_init()

```
int wifi_init (
    const wifi_init_config_t * config,
    wifi_init_complete_cb_t init_cb )
```

Init Wi-Fi Initializes the wifi according to the specified parameters in the config.

Attention

1. This API must be called before other Wi-Fi APIs are invoked

Parameters

in	<i>config</i>	pointer to Wi-Fi init configuration structure; can point to a temporary variable.
in	<i>init_cb</i>	pointer to Wi-Fi init complete configuration structure; can point to a temporary variable.

Returns

0 : success
other : failed

4.9.3.41 wifi_scan_get_ap_list()

```
int wifi_scan_get_ap_list (
    wifi_scan_list_t * scan_list )
```

Get list of APs that found in last scan operation.

Attention

This API only be called when scan is completed, otherwise it may get wrong value.

Parameters

out	<i>scan_list</i>	store APs' informaton that found in last scan operation
-----	------------------	---

Returns

0 : success
other : failed

4.9.3.42 wifi_scan_get_ap_num()

```
int wifi_scan_get_ap_num (
    uint16_t * number )
```

Get the number of scanned APs.

Parameters

out	<i>number</i>	store number of APs found in last scan operation
-----	---------------	--

Attention

This API only be called when scan is completed, otherwise it may get wrong value.

Returns

the scan result of AP number

4.9.3.43 wifi_scan_get_ap_records()

```
int wifi_scan_get_ap_records (
    uint16_t * number,
    wifi_scan_info_t * ap_records )
```

Get AP list found in last scan operation.

Parameters

out	<i>number</i>	As input param, it stores max AP number that ap_records can hold. As output param, it receives the actual AP number that this API returns.
out	<i>ap_records</i>	wifi_scan_info_t array stores the found APs

Returns

0 : success
other : failed

4.9.3.44 wifi_scan_scan_stop()

```
int wifi_scan_scan_stop (
    void )
```

Stop scanning process.

Attention

This API shall be called after [wifi_scan_start\(\)](#)

Returns

0 : success
other : failed

4.9.3.45 wifi_scan_start()

```
int wifi_scan_start (
    const wifi\_scan\_config\_t * config,
    bool block )
```

Scan all available APs. After invoke the [wifi_set_config\(\)](#) and [wifi_start\(\)](#), then call [wifi_scan_start\(\)](#) to scan APs.

Parameters

in	<i>config</i>	Configure parameters for scan operation
in	<i>block</i>	if block is true, this API blocks the caller until scan operation is done, otherwise it returns immediately

Returns

0 : success
other : failed

4.9.3.46 wifi_set_config()

```
int wifi_set_config (
    wifi\_mode\_t interface,
    wifi\_config\_t * conf )
```


Set configuration of OPL1000 STA.

Attention

1. This API is called only when specified interface is enabled, otherwise API calling will be failed
2. For station configuration, bssid_set shall be set to 0; set to 1 means user want to check MAC address of certain AP.
3. OPL1000 is limited to working on one channel.

Parameters

in	<i>interface</i>	Configure wifi working mode, The options are <ul style="list-style-type: none">• WIFI_MODE_STA• WIFI_MODE_AP (currently not support)
in	<i>conf</i>	structure of configuration paremeters

Returns

0 : success
other : failed

4.9.3.47 wifi_sta_get_ap_info()

```
int wifi_sta_get_ap_info (
    wifi_scan_info_t * ap_info )
```

Get information of AP which OPL1000 station is associated with.

Parameters

out	<i>ap_info</i>	get AP information from list
-----	----------------	------------------------------

Returns

0 : success
other : failed

4.9.3.48 wifi_start()

```
int wifi_start (
    void )
```

Start Wi-Fi working.

- If mode is WIFI_MODE_STA, it creates station control block and starts station

Returns

0 : success
other : failed

4.9.3.49 wifi_stop()

```
int wifi_stop (  
    void )
```

Stop wifi working.

- If mode is WIFI_MODE_STA, it stops station and releases station control block

Returns

0 : success
other : failed

4.10 Enumeration

Enumerations

- enum `wifi_auth_mode_t` {
`WIFI_AUTH_OPEN` = 0, `WIFI_AUTH_WEP`, `WIFI_AUTH_WPA_PSK`, `WIFI_AUTH_WPA2_PSK`,
`WIFI_AUTH_WPA_WPA2_PSK`, `WIFI_AUTH_WPA2_ENTERPRISE` }
This enumeration defines the wireless authentication mode to indicate the Wi-Fi device authentication attribute.
- enum `wifi_bandwidth_t` { `WIFI_BW_HT20` = 1, `WIFI_BW_HT40` }
- enum `wifi_cipher_type_t` {
`WIFI_CIPHER_TYPE_NONE` = 0, `WIFI_CIPHER_TYPE_WEP40`, `WIFI_CIPHER_TYPE_WEP104`,
`WIFI_CIPHER_TYPE_TKIP`,
`WIFI_CIPHER_TYPE_CCMP`, `WIFI_CIPHER_TYPE_TKIP_CCMP`, `WIFI_CIPHER_TYPE_UNKNOWN` }
This enumeration defines wireless security cipher suits.
- enum `wifi_event_t` {
`WIFI_EVENT_NONE` = -1, `WIFI_EVENT_INIT_COMPLETE` = 0, `WIFI_EVENT_SCAN_COMPLETE`,
`WIFI_EVENT_STA_START`,
`WIFI_EVENT_STA_STOP`, `WIFI_EVENT_STA_CONNECTED`, `WIFI_EVENT_STA_DISCONNECTED`,
`WIFI_EVENT_STA_CONNECTION_FAILED`,
`WIFI_EVENT_STA_GOT_IP`, `WIFI_EVENT_MAX` }
This enumeration defines the supported events generated by the Wi-Fi driver. The event will be sent to the upper layer handler registered in `wifi_register_event_handler()`.
- enum `wifi_mode_t` { `WIFI_MODE_NULL` = 0, `WIFI_MODE_STA`, `WIFI_MODE_AP`, `WIFI_MODE_MAX` }
- enum `wifi_reason_code_t` {
`WIFI_REASON_CODE_SUCCESS`, `WIFI_REASON_CODE_FIND_AP_FAIL`,
`WIFI_REASON_CODE_PREV_AUTH_INVALID`,
`WIFI_REASON_CODE_DEAUTH_LEAVING_BSS`,
`WIFI_REASON_CODE_DISASSOC_INACTIVITY`, `WIFI_REASON_CODE_DISASSOC_AP_OVERLOAD`,
`WIFI_REASON_CODE_CLASS_2_ERR`, `WIFI_REASON_CODE_CLASS_3_ERR`,
`WIFI_REASON_CODE_DISASSOC_LEAVING_BSS`, `WIFI_REASON_CODE_ASSOC_BEFORE_AUTH`,
`WIFI_REASON_CODE_DISASSOC_PWR_CAP_UNACCEPTABLE`,
`WIFI_REASON_CODE_DISASSOC_SUP_CHS_UNACCEPTABLE`, `WIFI_REASON_CODE_INVALID_INFO_ELEM` = 13,
`WIFI_REASON_CODE_MIC_FAILURE`, `WIFI_REASON_CODE_4_WAY_HANDSHAKE_TIMEOUT`,
`WIFI_REASON_CODE_GROUP_KEY_UPDATE_TIMEOUT`,
`WIFI_REASON_CODE_DIFFERENT_INFO_ELEM`, `WIFI_REASON_CODE_GROUP_CIPHER_INVALID_VALID`,
`WIFI_REASON_CODE_PAIRWISE_CIPHER_INVALID`, `WIFI_REASON_CODE_AKMP_INVALID`,
`WIFI_REASON_CODE_UNSUPPORTED_RSNE_VERSION`,
`WIFI_REASON_CODE_INVALID_RSNE_CAPABILITIES`,
`WIFI_REASON_CODE_IEEE_802_1X_AUTH_FAILED`, `WIFI_REASON_CODE_CIPHER_REJECTED`,
`WIFI_REASON_CODE_AUTO_CONNECT_FAILED` = 200, `WIFI_REASON_CODE_CONNECT_NOT_FOUND`,
`WIFI_REASON_CODE_CONNECT_TIMEOUT` }
This enumeration defines the reason code of the `WIFI_EVENT_STA_CONNECTION_FAILED` event in `wifi_event_t`. Find the details for the reason code below.
- enum `wifi_scan_method_t` { `WIFI_FAST_SCAN` = 0, `WIFI_ALL_CHANNEL_SCAN` }
- enum `wifi_scan_type_t` { `WIFI_SCAN_TYPE_ACTIVE` = 0, `WIFI_SCAN_TYPE_PASSIVE` }
This enumeration defines the wireless STA scan type.
- enum `wifi_sort_method_t` { `WIFI_CONNECT_AP_BY_SIGNAL` = 0, `WIFI_CONNECT_AP_BY_SECURITY` }

4.10.1 Detailed Description

4.10.2 Enumeration Type Documentation

4.10.2.1 `wifi_auth_mode_t`

```
enum wifi_auth_mode_t
```

This enumeration defines the wireless authentication mode to indicate the Wi-Fi device authentication attribute.

Enumerator

WIFI_AUTH_OPEN	authenticate mode : open
WIFI_AUTH_WEP	authenticate mode : WEP
WIFI_AUTH_WPA_PSK	authenticate mode : WPA_PSK
WIFI_AUTH_WPA2_PSK	authenticate mode : WPA2_PSK
WIFI_AUTH_WPA_WPA2_PSK	authenticate mode : WPA_WPA2_PSK
WIFI_AUTH_WPA2_ENTERPRISE	authenticate mode : WPA2_ENTERPRISE

4.10.2.2 wifi_bandwidth_t

```
enum wifi_bandwidth_t
```

Enumerator

WIFI_BW_HT20	Bandwidth is HT20
WIFI_BW_HT40	Bandwidth is HT40

4.10.2.3 wifi_cipher_type_t

```
enum wifi_cipher_type_t
```

This enumeration defines wireless security cipher suits.

Enumerator

WIFI_CIPHER_TYPE_NONE	0, the cipher type is none
WIFI_CIPHER_TYPE_WEP40	1, the cipher type is WEP40
WIFI_CIPHER_TYPE_WEP104	2, the cipher type is WEP104
WIFI_CIPHER_TYPE_TKIP	3, the cipher type is TKIP
WIFI_CIPHER_TYPE_CCMP	4, the cipher type is CCMP
WIFI_CIPHER_TYPE_TKIP_CCMP	5, the cipher type is TKIP and CCMP
WIFI_CIPHER_TYPE_UNKNOWN	6, the cipher type is unknown

4.10.2.4 wifi_event_t

```
enum wifi_event_t
```

This enumeration defines the supported events generated by the Wi-Fi driver. The event will be sent to the upper layer handler registered in [wifi_register_event_handler\(\)](#).

Enumerator

WIFI_EVENT_NONE	Reserved
WIFI_EVENT_INIT_COMPLETE	Wi-Fi initialization complete event.
WIFI_EVENT_SCAN_COMPLETE	Scan completed event
WIFI_EVENT_STA_START	station start
WIFI_EVENT_STA_STOP	station stop
WIFI_EVENT_STA_CONNECTED	station connected to AP event
WIFI_EVENT_STA_DISCONNECTED	station disconnected from AP
WIFI_EVENT_STA_CONNECTION_FAILED	Connection has failed. For the reason code, please refer to wifi_reason_code_t .
WIFI_EVENT_STA_GOT_IP	station got IP from connected AP
WIFI_EVENT_MAX	

4.10.2.5 wifi_mode_t

```
enum wifi_mode_t
```

Enumerator

WIFI_MODE_NULL	null mode
WIFI_MODE_STA	Wi-Fi station mode
WIFI_MODE_AP	Wi-Fi soft-AP mode
WIFI_MODE_MAX	

4.10.2.6 wifi_reason_code_t

```
enum wifi_reason_code_t
```

This enumeration defines the reason code of the WIFI_EVENT_STA_CONNECTION_FAILED event in [wifi_event_t](#). Find the details for the reason code below.

Enumerator

WIFI_REASON_CODE_SUCCESS	0 Reserved.
WIFI_REASON_CODE_FIND_AP_FAIL	1 (Internal) No AP found.
WIFI_REASON_CODE_PREV_AUTH_INVALID	2 Previous authentication is no longer valid.
WIFI_REASON_CODE_DEAUTH_LEAVING_BSS	3 Deauthenticated because sending STA is leaving (or has left) IBSS or ES.
WIFI_REASON_CODE_DISASSOC_INACTIVITY	4 Disassociated due to inactivity.
WIFI_REASON_CODE_DISASSOC_AP_OVERLOAD	5 Disassociated because AP is unable to handle all currently associated STAs.
WIFI_REASON_CODE_CLASS_2_ERR	6 Class 2 frame received from nonauthenticated STA.
WIFI_REASON_CODE_CLASS_3_ERR	7 Class 3 frame received from nonauthenticated STA.

Enumerator

WIFI_REASON_CODE_DISASSOC_LEAVING_BSS	8 Disassociated because sending STA is leaving (or has left) BSS.
WIFI_REASON_CODE_ASSOC_BEFORE_AUTH	9 STA requesting (re)association is not authenticated with responding STA.
WIFI_REASON_CODE_DISASSOC_PWR_CAP_↔ UNACCEPTABLE	10 Disassociated because the information in the Power Capability element is unacceptable.
WIFI_REASON_CODE_DISASSOC_SUP_CHS_U↔ NACCEPTABLE	11 Disassociated because the information in the Supported Channels element is unacceptable.
WIFI_REASON_CODE_INVALID_INFO_ELEM	13 Invalid information element.
WIFI_REASON_CODE_MIC_FAILURE	14 Message integrity code (MIC) failure.
WIFI_REASON_CODE_4_WAY_HANDSHAKE_TI↔ MEOUT	15 4-Way Handshake time out.
WIFI_REASON_CODE_GROUP_KEY_UPDATE_↔ TIMEOUT	16 Group Key Handshake time out.
WIFI_REASON_CODE_DIFFERENT_INFO_ELEM	17 Information element in 4-Way Handshake different from (Re)Association Request/Probe Response/Beacon frame.
WIFI_REASON_CODE_GROUP_CIPHER_INVALID↔ D_VALID	18 Invalid group cipher.
WIFI_REASON_CODE_PAIRWISE_CIPHER_INV↔ ALID	19 Invalid pairwise cipher.
WIFI_REASON_CODE_AKMP_INVALID	20 Invalid AKMP.
WIFI_REASON_CODE_UNSUPPORTED_RSNE_↔ VERSION	21 Unsupported RSN information element version.
WIFI_REASON_CODE_INVALID_RSNE_CAPABI↔ LITIES	22 Invalid RSN information element capabilities.
WIFI_REASON_CODE_IEEE_802_1X_AUTH_FAI↔ LED	23 IEEE 802.1X authentication failed.
WIFI_REASON_CODE_CIPHER_REJECTED	24 Cipher suite rejected because of the security policy.
WIFI_REASON_CODE_AUTO_CONNECT_FAILED	200 Auto connect failed.
WIFI_REASON_CODE_CONNECT_NOT_FOUND	201 The target AP is not found.
WIFI_REASON_CODE_CONNECT_TIMEOUT	202 Connect to AP timeout.

4.10.2.7 wifi_scan_method_t

```
enum wifi_scan_method_t
```

Enumerator

WIFI_FAST_SCAN	Do fast scan, scan will end after find SSID match AP
WIFI_ALL_CHANNEL_SCAN	All channel scan, scan will end after scan all the channel

4.10.2.8 wifi_scan_type_t

```
enum wifi_scan_type_t
```

This enumeration defines the wireless STA scan type.

Enumerator

WIFI_SCAN_TYPE_ACTIVE	Actively scan a network by sending 802.11 probe(s)
WIFI_SCAN_TYPE_PASSIVE	Passively scan a network by listening for beacons from APs

4.10.2.9 wifi_sort_method_t

```
enum wifi_sort_method_t
```

Enumerator

WIFI_CONNECT_AP_BY_SIGNAL	Sort match AP in scan list by RSSI
WIFI_CONNECT_AP_BY_SECURITY	Sort match AP in scan list by security mode

Chapter 5

Data Structure Documentation

5.1 auto_conn_info_t Struct Reference

```
#include <controller_wifi_com_patch.h>
```

Data Fields

- u8 [ap_channel](#)
- u16 [beacon_interval](#)
- u8 [bssid](#) [MAC_ADDR_LEN]
- u16 [capabilities](#)
- u8 [dtim_prod](#)
- u8 [fast_connect](#)
- bool [free_ocpy](#)
- s8 [hid_ssid](#) [IEEE80211_MAX_SSID_LEN+1]
- u64 [latest_beacon_rx_time](#)
- s8 [passphrase](#) [MAX_LEN_OF_PASSPHRASE]
- u8 [psk](#) [32]
- u8 [rsn_ie](#) [100]
- s8 [rssi](#)
- s8 [ssid](#) [IEEE80211_MAX_SSID_LEN+1]
- u8 [supported_rates](#) [SUPPORTED_RATES_MAX]
- wpa_ie_data_t [wpa_data](#)
- u8 [wpa_ie](#) [100]

5.1.1 Field Documentation

5.1.1.1 ap_channel

u8 [ap_channel](#)

5.1.1.2 beacon_interval

u16 beacon_interval

5.1.1.3 bssid

u8 bssid[MAC_ADDR_LEN]

5.1.1.4 capabilities

u16 capabilities

5.1.1.5 dtim_prod

u8 dtim_prod

5.1.1.6 fast_connect

u8 fast_connect

5.1.1.7 free_ocpy

bool free_ocpy

5.1.1.8 hid_ssid

s8 hid_ssid[IEEE80211_MAX_SSID_LEN+1]

5.1.1.9 latest_beacon_rx_time

u64 latest_beacon_rx_time

5.1.1.10 passphrase

```
s8 passphrase[MAX_LEN_OF_PASSPHRASE]
```

5.1.1.11 psk

```
u8 psk[32]
```

5.1.1.12 rsn_ie

```
u8 rsn_ie[100]
```

5.1.1.13 rssi

```
s8 rssi
```

5.1.1.14 ssid

```
s8 ssid[IEEE80211_MAX_SSID_LEN+1]
```

5.1.1.15 supported_rates

```
u8 supported_rates[SUPPORTED_RATES_MAX]
```

5.1.1.16 wpa_data

```
wpa_ie_data_t wpa_data
```

5.1.1.17 wpa_ie

```
u8 wpa_ie[100]
```

5.2 auto_connect_cfg_t Struct Reference

```
#include <controller_wifi_com_patch.h>
```

Data Fields

- bool [flag](#)
- s8 [front](#)
- u8 [max_save_num](#)
- [auto_conn_info_t](#) * [pFCInfo](#)
- s8 [rear](#)
- u8 [retryCount](#)
- u8 [targetIdx](#)
- u32 [uFCApNum](#)

5.2.1 Field Documentation

5.2.1.1 flag

```
bool flag
```

5.2.1.2 front

```
s8 front
```

5.2.1.3 max_save_num

```
u8 max_save_num
```

5.2.1.4 pFCInfo

```
auto\_conn\_info\_t* pFCInfo
```

5.2.1.5 rear

s8 rear

5.2.1.6 retryCount

u8 retryCount

5.2.1.7 targetIdx

u8 targetIdx

5.2.1.8 uFCapNum

u32 uFCapNum

5.3 event_msg_t Struct Reference

Send information to event by [event_msg_t](#).

```
#include <event_loop.h>
```

Data Fields

- uint32_t [event](#)
- uint32_t [length](#)
- uint8_t * [param](#)

5.3.1 Detailed Description

Send information to event by [event_msg_t](#).

5.3.2 Field Documentation

5.3.2.1 event

```
uint32_t event
```

event type

5.3.2.2 length

```
uint32_t length
```

Packet length

5.3.2.3 param

```
uint8_t* param
```

event paramment

5.4 hap_control_t Struct Reference

```
#include <controller_wifi_com_patch.h>
```

Data Fields

- [auto_conn_info_t](#) * [hap_ap_info](#)
- int [hap_bitvector](#)
- u8 [hap_en](#)
- int [hap_final_index](#)
- int [hap_index](#)
- char [hap_ssid](#) [IEEE80211_MAX_SSID_LEN+1]

5.4.1 Field Documentation

5.4.1.1 hap_ap_info

```
auto\_conn\_info\_t* hap_ap_info
```

5.4.1.2 hap_bitvector

```
int hap_bitvector
```

5.4.1.3 hap_en

u8 hap_en

5.4.1.4 hap_final_index

int hap_final_index

5.4.1.5 hap_index

int hap_index

5.4.1.6 hap_ssid

char hap_ssid[IEEE80211_MAX_SSID_LEN+1]

5.5 LE_BT_ADDR_T Struct Reference

```
#include <ble.h>
```

Data Fields

- BD_ADDR [addr](#)
- UINT8 [type](#)

5.5.1 Field Documentation

5.5.1.1 addr

BD_ADDR addr

address

5.5.1.2 type

UINT8 type

address type

5.6 LE_CM_CONNECTION_COMPLETE_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [conn_interval](#)
- UINT16 [conn_latency](#)
- UINT16 [dev_id](#)
- BD_ADDR [peer_addr](#)
- UINT8 [peer_addr_type](#)
- UINT8 [role](#)
- UINT16 [status](#)
- UINT16 [supervison_timeout](#)

5.6.1 Field Documentation

5.6.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.6.1.2 conn_interval

UINT16 conn_interval

connection interval

5.6.1.3 conn_latency

UINT16 conn_latency

connection latency

5.6.1.4 dev_id

UINT16 dev_id

device ID

5.6.1.5 peer_addr

BD_ADDR peer_addr

peer address

5.6.1.6 peer_addr_type

UINT8 peer_addr_type

peer address type

5.6.1.7 role

UINT8 role

master or slave

5.6.1.8 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.6.1.9 supervision_timeout

UINT16 supervision_timeout

supervision timeout

5.7 LE_CM_MSG_ADVERTISE_REPORT_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- BD_ADDR [addr](#)
- UINT8 [addr_type](#)
- UINT8 [data](#) [1]
- UINT8 [event_type](#)
- UINT8 [len](#)
- INT8 [rssi](#)

5.7.1 Field Documentation

5.7.1.1 addr

BD_ADDR addr

address

5.7.1.2 addr_type

UINT8 addr_type

address type

5.7.1.3 data

UINT8 data[1]

5.7.1.4 event_type

UINT8 event_type

5.7.1.5 len

UINT8 len

5.7.1.6 rssi

INT8 rssi

RSSI

5.8 LE_CM_MSG_CONN_PARA_REQ_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [itv_max](#)
- UINT16 [itv_min](#)
- UINT16 [latency](#)
- UINT32 [sv_tmo](#)

5.8.1 Field Documentation

5.8.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.8.1.2 itv_max

UINT16 itv_max

maximum connection interval

5.8.1.3 itv_min

UINT16 itv_min

minimum connection interval

5.8.1.4 latency

UINT16 latency

slave latency

5.8.1.5 sv_tmo

UINT32 sv_tmo

supervision timeout

5.9 LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [interval](#)
- UINT16 [latency](#)
- UINT16 [status](#)
- UINT32 [supervision_timeout](#)

5.9.1 Field Documentation

5.9.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.9.1.2 interval

UINT16 interval

connection interval

5.9.1.3 latency

UINT16 latency

slave letency

5.9.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.9.1.5 supervision_timeout

UINT32 supervision_timeout

supervision timeout

5.10 LE_CM_MSG_DATA_LEN_CHANGE_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [max_rx_octets](#)
- UINT16 [max_rx_time](#)
- UINT16 [max_tx_octets](#)
- UINT16 [max_tx_time](#)

5.10.1 Field Documentation

5.10.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.10.1.2 max_rx_octets

UINT16 max_rx_octets

connMaxRxOctets

5.10.1.3 max_rx_time

UINT16 max_rx_time

connMaxRxTime

5.10.1.4 max_tx_octets

UINT16 max_tx_octets

connMaxTxOctets

5.10.1.5 max_tx_time

UINT16 max_tx_time

connMaxTxTime

5.11 LE_CM_MSG_DIRECT_ADV_REPORT_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- BD_ADDR [direct_addr](#)
- UINT8 [direct_addr_type](#)
- BD_ADDR [peer_addr](#)
- UINT8 [peer_addr_type](#)
- INT8 [rssi](#)

5.11.1 Field Documentation

5.11.1.1 `direct_addr`

BD_ADDR `direct_addr`

direct address

5.11.1.2 `direct_addr_type`

UINT8 `direct_addr_type`

direct address type

5.11.1.3 `peer_addr`

BD_ADDR `peer_addr`

peer address

5.11.1.4 `peer_addr_type`

UINT8 `peer_addr_type`

peer address type

5.11.1.5 `rssi`

INT8 `rssi`

RSSI

5.12 LE_CM_MSG_DISCONNECT_COMPLETE_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT8 [reason](#)
- UINT16 [status](#)

5.12.1 Field Documentation

5.12.1.1 conn_hdl

UINT16 [conn_hdl](#)

connection handle

5.12.1.2 reason

UINT8 [reason](#)

disconnect reason

5.12.1.3 status

UINT16 [status](#)

refer to LE_ERR_STATE in [ble_err.h](#)

5.13 LE_CM_MSG_ENCRYPTION_CHANGE_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT8 [enabled](#)
- UINT16 [status](#)

5.13.1 Field Documentation

5.13.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.13.1.2 devid

UINT16 devid

device ID

5.13.1.3 enabled

UINT8 enabled

5.13.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.14 LE_CM_MSG_ENCRYPTION_REFRESH_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- BOOL [enabled](#)
- UINT16 [status](#)

5.14.1 Field Documentation

5.14.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.14.1.2 devid

UINT16 devid

device ID

5.14.1.3 enabled

BOOL enabled

enable or disable

5.14.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.15 LE_CM_MSG_INIT_COMPLETE_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [status](#)

5.15.1 Field Documentation

5.15.1.1 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.16 LE_CM_MSG_LTK_REQ_IND_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [ediv](#)
- UINT8 [rand](#) [8]

5.16.1 Field Documentation

5.16.1.1 conn_hdl

UINT16 [conn_hdl](#)

connection handle

5.16.1.2 devid

UINT16 [devid](#)

device ID

5.16.1.3 ediv

UINT16 [ediv](#)

5.16.1.4 rand

UINT8 [rand](#)[8]

5.17 LE_CM_MSG_READ_ADV_TX_POWER_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- INT8 [pwr_level](#)
- UINT16 [status](#)

5.17.1 Field Documentation

5.17.1.1 pwr_level

INT8 pwr_level

power level

5.17.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.18 LE_CM_MSG_READ_BD_ADDR_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- BD_ADDR [bd_addr](#)
- UINT16 [status](#)

5.18.1 Field Documentation

5.18.1.1 bd_addr

BD_ADDR bd_addr

5.18.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.19 LE_CM_MSG_READ_CHANNEL_MAP_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT8 [ch_map](#) [5]
- UINT16 [conn_hdl](#)
- UINT16 [status](#)

5.19.1 Field Documentation

5.19.1.1 [ch_map](#)

```
UINT8 ch_map[5]
```

channel map

5.19.1.2 [conn_hdl](#)

```
UINT16 conn_hdl
```

connection handle

5.19.1.3 [status](#)

```
UINT16 status
```

refer to LE_ERR_STATE in [ble_err.h](#)

5.20 LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT8 [size](#)
- UINT16 [status](#)

5.20.1 Field Documentation

5.20.1.1 size

UINT8 size

resolving list size

5.20.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.21 LE_CM_MSG_READ_RSSI_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- INT8 [rssi](#)
- UINT16 [status](#)

5.21.1 Field Documentation

5.21.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.21.1.2 rssi

INT8 rssi

RSSI

5.21.1.3 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.22 LE_CM_MSG_READ_TX_POWER_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [status](#)
- INT8 [tx_power](#)

5.22.1 Field Documentation

5.22.1.1 [conn_hdl](#)

UINT16 [conn_hdl](#)

connection handle

5.22.1.2 [status](#)

UINT16 [status](#)

refer to LE_ERR_STATE in [ble_err.h](#)

5.22.1.3 [tx_power](#)

INT8 [tx_power](#)

tx power

5.23 LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT8 [size](#)
- UINT16 [status](#)

5.23.1 Field Documentation

5.23.1.1 size

UINT8 size

white list size

5.23.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.24 LE_CM_MSG_SET_DATA_LENGTH_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [status](#)

5.24.1 Field Documentation

5.24.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.24.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.25 LE_CM_MSG_SET_DISCONNECT_CFM_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [handle](#)
- UINT16 [status](#)

5.25.1 Field Documentation

5.25.1.1 handle

UINT16 handle

connection handle

5.25.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.26 LE_CM_MSG_SIGNAL_UPDATE_REQ_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [identifier](#)
- UINT16 [interval_max](#)
- UINT16 [interval_min](#)
- UINT16 [slave_latency](#)
- UINT32 [timeout_multiplier](#)

5.26.1 Field Documentation

5.26.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.26.1.2 identifier

UINT16 identifier

5.26.1.3 interval_max

UINT16 interval_max

maximum connection interval

5.26.1.4 interval_min

UINT16 interval_min

minimum connection interval

5.26.1.5 slave_latency

UINT16 slave_latency

slave latency

5.26.1.6 timeout_multiplier

UINT32 timeout_multiplier

5.27 LE_CM_REQ_STATUS_T Struct Reference

```
#include <ble_cm_if.h>
```

Data Fields

- UINT16 [status](#)

5.27.1 Field Documentation

5.27.1.1 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.28 LE_CONN_PARA_T Struct Reference

```
#include <ble.h>
```

Data Fields

- UINT16 [itv_max](#)
- UINT16 [itv_min](#)
- UINT16 [latency](#)
- UINT16 [sv_timeout](#)

5.28.1 Field Documentation

5.28.1.1 itv_max

UINT16 itv_max

maximum connection interval

5.28.1.2 itv_min

UINT16 itv_min

mininum connection interval

5.28.1.3 latency

UINT16 latency

slave latency

5.28.1.4 sv_timeout

UINT16 sv_timeout

supervision timeout

5.29 LE_GAP_ADVERTISING_PARAM_T Struct Reference

```
#include <ble_gap_if.h>
```

Data Fields

- UINT8 [channel_map](#)
- UINT8 [filter_policy](#)
- UINT16 [interval_max](#)
- UINT16 [interval_min](#)
- UINT8 [own_addr_type](#)
- BD_ADDR [peer_addr](#)
- UINT8 [peer_addr_type](#)
- UINT8 [type](#)

5.29.1 Field Documentation

5.29.1.1 [channel_map](#)

UINT8 [channel_map](#)

advertising channel map

5.29.1.2 [filter_policy](#)

UINT8 [filter_policy](#)

advertising filter policy

5.29.1.3 [interval_max](#)

UINT16 [interval_max](#)

maximum advertising interval

5.29.1.4 [interval_min](#)

UINT16 [interval_min](#)

minimum advertising interval

5.29.1.5 own_addr_type

UINT8 own_addr_type

owner address type

5.29.1.6 peer_addr

BD_ADDR peer_addr

peer address

5.29.1.7 peer_addr_type

UINT8 peer_addr_type

peer address type

5.29.1.8 type

UINT8 type

advertising type

5.30 LE_GAP_CONN_PARAM_T Struct Reference

```
#include <ble_gap_if.h>
```

Data Fields

- UINT16 [interval_max](#)
- UINT16 [interval_min](#)
- UINT16 [latency](#)
- UINT16 [supervision_timeout](#)

5.30.1 Field Documentation

5.30.1.1 interval_max

UINT16 interval_max

maximum connection interval

5.30.1.2 interval_min

UINT16 interval_min

minimum connection interval

5.30.1.3 latency

UINT16 latency

slave latency

5.30.1.4 supervision_timeout

UINT16 supervision_timeout

supervision timeout for the LE Link

5.31 LE_GAP_SCAN_PARAM_T Struct Reference

```
#include <ble_gap_if.h>
```

Data Fields

- [UINT8 filter_policy](#)
- [UINT16 interval](#)
- [UINT8 own_addr_type](#)
- [UINT8 type](#)
- [UINT16 window](#)

5.31.1 Field Documentation

5.31.1.1 filter_policy

UINT8 filter_policy

scan filter policy

5.31.1.2 interval

UINT16 interval

scan interval

5.31.1.3 own_addr_type

UINT8 own_addr_type

owner address type

5.31.1.4 type

UINT8 type

scan type

5.31.1.5 window

UINT16 window

scan window

5.32 LE_GATT_ATTR_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [format](#)
- UINT16 [handle](#)
- UINT16 [len](#)
- UINT16 [maxLen](#)
- UINT16 [permit](#)
- UINT16 *const [pUuid](#)
- UINT8 *const [pVal](#)

5.32.1 Field Documentation

5.32.1.1 format

UINT8 format

UUID type

5.32.1.2 handle

UINT16 handle

handle

5.32.1.3 len

UINT16 len

value length

5.32.1.4 maxLen

UINT16 maxLen

maximum value length

5.32.1.5 permit

UINT16 permit

permit

5.32.1.6 pUuid

UINT16* const pUuid

UUID

5.32.1.7 pVal

UINT8* const pVal

value

5.33 LE_GATT_MSG_ACCESS_READ_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [offset](#)

5.33.1 Field Documentation

5.33.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.33.1.2 devid

UINT16 devid

device index

5.33.1.3 handle

UINT16 handle

attribute handle

5.33.1.4 offset

UINT16 offset

attribute handle value

5.34 LE_GATT_MSG_ACCESS_WRITE_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT8 [flag](#)
- UINT16 [handle](#)
- UINT16 [len](#)
- UINT16 [offset](#)
- UINT8 * [pVal](#)

5.34.1 Field Documentation

5.34.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.34.1.2 devid

UINT16 devid

device ID

5.34.1.3 flag

UINT8 flag

refer to LE_GATT_FLAG_* in [ble_gatt_if.h](#)

5.34.1.4 handle

UINT16 handle

attribute handle

5.34.1.5 len

UINT16 len

length written

5.34.1.6 offset

UINT16 offset

attribute handle value

5.34.1.7 pVal

UINT8* pVal

value written

5.35 LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT8 [format](#)
- UINT16 [handle](#)
- UINT16 [uuid](#) [8]

5.35.1 Field Documentation

5.35.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.35.1.2 devid

UINT16 devid

device ID

5.35.1.3 format

UINT8 format

UUID type

5.35.1.4 handle

UINT16 handle

characteristic descriptor handle

5.35.1.5 uuid

UINT16 uuid[8]

UUID

5.36 LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT8 [format](#)
- UINT16 [handle](#)
- UINT8 [property](#)
- UINT16 [uuid](#) [8]
- UINT16 [val_hdl](#)

5.36.1 Field Documentation

5.36.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.36.1.2 devid

UINT16 devid

device ID

5.36.1.3 format

UINT8 format

UUID type

5.36.1.4 handle

UINT16 handle

characteristic declaration handle

5.36.1.5 property

UINT8 property

property

5.36.1.6 uuid

```
UINT16 uuid[8]
```

UUID

5.36.1.7 val_hdl

```
UINT16 val_hdl
```

characteristic value handle

5.37 LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [len](#)
- UINT16 [offset](#)
- UINT8 * [val](#)

5.37.1 Field Documentation

5.37.1.1 att_err

```
UINT8 att_err
```

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.37.1.2 conn_hdl

```
UINT16 conn_hdl
```

connection handle

5.37.1.3 devid

UINT16 devid

device ID

5.37.1.4 handle

UINT16 handle

characteristic value handle

5.37.1.5 len

UINT16 len

value length

5.37.1.6 offset

UINT16 offset

value position offset

5.37.1.7 val

UINT8* val

value

5.38 LE_GATT_MSG_CONFIRMATION_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)

5.38.1 Field Documentation

5.38.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.38.1.2 devid

UINT16 devid

device ID

5.38.1.3 handle

UINT16 handle

attribute handle

5.39 LE_GATT_MSG_EXCHANGE_MTU_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [current_rx_mtu](#)
- UINT16 [devid](#)

5.39.1 Field Documentation

5.39.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.39.1.2 current_rx_mtu

UINT16 current_rx_mtu

current receive MTU

5.39.1.3 devid

UINT16 devid

device ID

5.40 LE_GATT_MSG_EXCHANGE_MTU_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [client_rx_mtu](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)

5.40.1 Field Documentation

5.40.1.1 client_rx_mtu

UINT16 client_rx_mtu

client receive MTU

5.40.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.40.1.3 devid

UINT16 devid

device ID

5.41 LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [err_hdl](#)
- UINT16 [status](#)

5.41.1 Field Documentation

5.41.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.41.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.41.1.3 devid

UINT16 devid

device ID

5.41.1.4 err_hdl

UINT16 err_hdl

TBD

5.41.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.42 LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```


Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.42.1 Field Documentation

5.42.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.42.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.42.1.3 devid

UINT16 devid

device ID

5.42.1.4 handle

UINT16 handle

characteristic descriptor handle

5.42.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.43 LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.43.1 Field Documentation

5.43.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.43.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.43.1.3 devid

UINT16 devid

device ID

5.43.1.4 handle

UINT16 handle

5.43.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.44 LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.44.1 Field Documentation

5.44.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.44.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.44.1.3 devid

UINT16 devid

device ID

5.44.1.4 handle

UINT16 handle

characteristic descriptor handle

5.44.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.45 LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.45.1 Field Documentation

5.45.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.45.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.45.1.3 devid

UINT16 devid

device ID

5.45.1.4 handle

UINT16 handle

include service start handle

5.45.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.46 LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.46.1 Field Documentation

5.46.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.46.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.46.1.3 devid

UINT16 devid

device ID

5.46.1.4 handle

UINT16 handle

service start handle

5.46.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.47 LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [end_hdl](#)
- UINT8 [format](#)
- UINT16 [handle](#)
- UINT16 [start_hdl](#)
- UINT16 [uuid](#) [8]

5.47.1 Field Documentation

5.47.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.47.1.2 devid

UINT16 devid

device ID

5.47.1.3 end_hdl

UINT16 end_hdl

end handle

5.47.1.4 format

UINT8 format

UUID type

5.47.1.5 handle

UINT16 handle

include servie handle

5.47.1.6 start_hdl

UINT16 start_hdl

start handle

5.47.1.7 uuid

UINT16 uuid[8]

UUID

5.48 LE_GATT_MSG_INDICATE_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [len](#)
- UINT8 * [val](#)

5.48.1 Field Documentation

5.48.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.48.1.2 devid

UINT16 devid

device ID

5.48.1.3 handle

UINT16 handle

attribute handle

5.48.1.4 len

UINT16 len

value length

5.48.1.5 val

UINT8* val

value

5.49 LE_GATT_MSG_NOTIFY_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.49.1 Field Documentation

5.49.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.49.1.2 devid

UINT16 devid

device ID

5.49.1.3 handle

UINT16 handle

attribute handle

5.49.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.50 LE_GATT_MSG_NOTIFY_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [len](#)
- UINT8 * [val](#)

5.50.1 Field Documentation

5.50.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.50.1.2 devid

UINT16 devid

device ID

5.50.1.3 handle

UINT16 handle

attribute handle

5.50.1.4 len

UINT16 len

value length

5.50.1.5 val

UINT8* val

value

5.51 LE_GATT_MSG_OPERATION_TIMEOUT_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_op](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)

5.51.1 Field Documentation

5.51.1.1 att_op

UINT8 att_op

refer to LE_ATT_OP_* in [ble_att_if.h](#)

5.51.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.51.1.3 devid

UINT16 devid

device ID

5.52 LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.52.1 Field Documentation

5.52.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.52.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.52.1.3 devid

UINT16 devid

device ID

5.52.1.4 handle

UINT16 handle

attribute handle

5.52.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.53 LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.53.1 Field Documentation

5.53.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.53.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.53.1.3 devid

UINT16 devid

device ID

5.53.1.4 handle

UINT16 handle

characteristic value handle

5.53.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.54 LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.54.1 Field Documentation

5.54.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.54.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.54.1.3 devid

UINT16 devid

device ID

5.54.1.4 handle

UINT16 handle

characteristic value handle

5.54.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.55 LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.55.1 Field Documentation

5.55.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.55.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.55.1.3 devid

UINT16 devid

device ID

5.55.1.4 handle

UINT16 handle

characteristic value handle

5.55.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.56 LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [err_hdl](#)
- UINT16 [len](#)
- UINT16 [status](#)
- UINT8 * [val](#)

5.56.1 Field Documentation

5.56.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.56.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.56.1.3 devid

UINT16 devid

device ID

5.56.1.4 err_hdl

UINT16 err_hdl

TBD

5.56.1.5 len

UINT16 len

value length

5.56.1.6 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.56.1.7 val

UINT8* val

value

5.57 LE_GATT_MSG_SERVICE_INFO_IND_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [end_hdl](#)
- UINT8 [format](#)
- UINT16 [start_hdl](#)
- UINT16 [uuid](#) [8]

5.57.1 Field Documentation

5.57.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.57.1.2 devid

UINT16 devid

device ID

5.57.1.3 end_hdl

UINT16 end_hdl

end handle

5.57.1.4 format

UINT8 format

UUID type

5.57.1.5 start_hdl

UINT16 start_hdl

start handle

5.57.1.6 uuid

UINT16 uuid[8]

UUID

5.58 LE_GATT_MSG_SIGNED_WRITE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.58.1 Field Documentation

5.58.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.58.1.2 devid

UINT16 devid

device ID

5.58.1.3 handle

UINT16 handle

attribute handle

5.58.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.59 LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.59.1 Field Documentation

5.59.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.59.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.59.1.3 devid

UINT16 devid

device ID

5.59.1.4 handle

UINT16 handle

characteristic value handle

5.59.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.60 LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.60.1 Field Documentation

5.60.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.60.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.60.1.3 devid

UINT16 devid

device ID

5.60.1.4 handle

UINT16 handle

attribute handle

5.60.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.61 LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT8 [att_err](#)
- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.61.1 Field Documentation

5.61.1.1 att_err

UINT8 att_err

0 is ok, others refer to LE_ATT_ERR_* in [ble_att_if.h](#)

5.61.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.61.1.3 devid

UINT16 devid

device ID

5.61.1.4 handle

UINT16 handle

characteristic value handle

5.61.1.5 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.62 LE_GATT_MSG_WRITE_NO_RSP_CFM_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [devid](#)
- UINT16 [handle](#)
- UINT16 [status](#)

5.62.1 Field Documentation

5.62.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.62.1.2 devid

UINT16 devid

device ID

5.62.1.3 handle

UINT16 handle

attribute handle

5.62.1.4 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.63 LE_GATT_SERVICE_T Struct Reference

```
#include <ble_gatt_if.h>
```

Data Fields

- UINT16 [endHdl](#)
- [LE_GATT_ATTR_T](#) * [pAttr](#)
- UINT16 [startHdl](#)
- UINT16 [svc_id](#)

5.63.1 Field Documentation

5.63.1.1 endHdl

UINT16 endHdl

end handle

5.63.1.2 pAttr

[LE_GATT_ATTR_T](#)* [pAttr](#)

pointer attribute table

5.63.1.3 startHdl

UINT16 startHdl

start handle

5.63.1.4 svc_id

UINT16 svc_id

service ID

5.64 LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- BOOL [enable](#)

5.64.1 Field Documentation

5.64.1.1 conn_hdl

UINT16 [conn_hdl](#)

connection handle

5.64.1.2 enable

BOOL [enable](#)

enable or disable

5.65 LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT16 [status](#)

5.65.1 Field Documentation

5.65.1.1 conn_hdl

UINT16 [conn_hdl](#)

connection handle

5.65.1.2 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.66 LE_SMP_MSG_OOB_DATA_REQUEST_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)

5.66.1 Field Documentation

5.66.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.67 LE_SMP_MSG_PAIRING_ACTION_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT8 [action](#)
- UINT16 [conn_hdl](#)
- BOOL [lost_bond](#)
- UINT8 [sc](#)

5.67.1 Field Documentation

5.67.1.1 action

UINT8 action

refer to LE_SM_IO_CAP_* in [ble_smp_if.h](#)

5.67.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.67.1.3 lost_bond

BOOL lost_bond

remote lost bond

5.67.1.4 sc

UINT8 sc

secure connection

5.68 LE_SMP_MSG_PAIRING_COMPLETE_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT8 [authenticated](#)
- UINT8 [bonded](#)
- UINT16 [conn_hdl](#)
- [LE_BT_ADDR_T](#) [peer_id_addr](#)
- UINT8 [sc](#)
- UINT16 [status](#)

5.68.1 Field Documentation

5.68.1.1 authenticated

UINT8 authenticated

authenticated

5.68.1.2 bonded

UINT8 bonded

bonded

5.68.1.3 conn_hdl

UINT16 conn_hdl

connection handle

5.68.1.4 peer_id_addr

[LE_BT_ADDR_T](#) peer_id_addr

peer device address

5.68.1.5 sc

UINT8 sc

secure connection

5.68.1.6 status

UINT16 status

refer to LE_ERR_STATE in [ble_err.h](#)

5.69 LE_SMP_MSG_PASSKEY_DISPLAY_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)
- UINT32 [passkey](#)

5.69.1 Field Documentation

5.69.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.69.1.2 passkey

UINT32 passkey

passkey

5.70 LE_SMP_MSG_PASSKEY_INPUT_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)

5.70.1 Field Documentation

5.70.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.71 LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT16 [conn_hdl](#)

5.71.1 Field Documentation

5.71.1.1 conn_hdl

UINT16 conn_hdl

connection handle

5.72 LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- [UINT8 bondable](#)
- [UINT16 conn_hdl](#)
- [UINT8 keypress](#)
- [UINT8 mitm](#)
- [UINT8 sc](#)

5.72.1 Field Documentation

5.72.1.1 bondable

UINT8 bondable

bonding

5.72.1.2 conn_hdl

UINT16 conn_hdl

connection handle

5.72.1.3 keypress

UINT8 keypress

keypress status

5.72.1.4 mitm

UINT8 mitm

MITM

5.72.1.5 sc

UINT8 sc

secure connection

5.73 LE_SMP_MSG_USER_CONFIRM_IND_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT32 [confirm_num](#)
- UINT16 [conn_hdl](#)

5.73.1 Field Documentation

5.73.1.1 confirm_num

UINT32 [confirm_num](#)

confirm number

5.73.1.2 conn_hdl

UINT16 [conn_hdl](#)

connection handle

5.74 LE_SMP_SC_OOB_DATA_T Struct Reference

```
#include <ble_smp_if.h>
```

Data Fields

- UINT8 [confirm](#) [16]
- UINT8 [rand](#) [16]

5.74.1 Field Documentation

5.74.1.1 confirm

UINT8 [confirm](#)[16]

confirm data

5.74.1.2 rand

```
UINT8 rand[16]
```

random data

5.75 LE_SYS_MSG_BUF_OVERFLOW_T Struct Reference

```
#include <ble_msg.h>
```

Data Fields

- UINT16 [conn_hdl](#)

5.75.1 Field Documentation

5.75.1.1 conn_hdl

```
UINT16 conn_hdl
```

connection handle

5.76 mw_wifi_auto_connect_ap_info_t Struct Reference

```
#include <controller_wifi_com_patch.h>
```

Data Fields

- u8 [ap_channel](#)
- u16 [beacon_interval](#)
- u8 [bssid](#) [MAC_ADDR_LEN]
- u16 [capabilities](#)
- u8 [dtim_prod](#)
- u8 [fast_connect](#)
- bool [free_ocpy](#)
- s8 [hid_ssid](#) [IEEE80211_MAX_SSID_LEN+1]
- u64 [latest_beacon_rx_time](#)
- s8 [passphrase](#) [64]
- u8 [psk](#) [32]
- u8 [rsn_ie](#) [100]
- s8 [rssi](#)
- s8 [ssid](#) [IEEE80211_MAX_SSID_LEN+1]
- u8 [supported_rates](#) [SUPPORTED_RATES_MAX]
- wpa_ie_data_t [wpa_data](#)
- u8 [wpa_ie](#) [100]

5.76.1 Field Documentation

5.76.1.1 ap_channel

u8 ap_channel

5.76.1.2 beacon_interval

u16 beacon_interval

5.76.1.3 bssid

u8 bssid[MAC_ADDR_LEN]

5.76.1.4 capabilities

u16 capabilities

5.76.1.5 dtim_prod

u8 dtim_prod

5.76.1.6 fast_connect

u8 fast_connect

5.76.1.7 free_ocpy

bool free_ocpy

5.76.1.8 hid_ssid

```
s8 hid_ssid[IEEE80211_MAX_SSID_LEN+1]
```

5.76.1.9 latest_beacon_rx_time

```
u64 latest_beacon_rx_time
```

5.76.1.10 passphrase

```
s8 passphrase[64]
```

5.76.1.11 psk

```
u8 psk[32]
```

5.76.1.12 rsn_ie

```
u8 rsn_ie[100]
```

5.76.1.13 rssi

```
s8 rssi
```

5.76.1.14 ssid

```
s8 ssid[IEEE80211_MAX_SSID_LEN+1]
```

5.76.1.15 supported_rates

```
u8 supported_rates[SUPPORTED_RATES_MAX]
```


5.76.1.16 wpa_data

```
wpa_ie_data_t wpa_data
```

5.76.1.17 wpa_ie

```
u8 wpa_ie[100]
```

5.77 MwFimAutoConnectCFG_t Struct Reference

```
#include <controller_wifi_com_patch.h>
```

Data Fields

- bool [flag](#)
- s8 [front](#)
- u8 [max_save_num](#)
- s8 [rear](#)
- u8 [targetIdx](#)

5.77.1 Field Documentation

5.77.1.1 flag

```
bool flag
```

5.77.1.2 front

```
s8 front
```

5.77.1.3 max_save_num

```
u8 max_save_num
```

5.77.1.4 rear

s8 rear

5.77.1.5 targetIdx

u8 targetIdx

5.78 T_RfCmd Struct Reference

```
#include <controller_wifi_patch.h>
```

Data Fields

- int [iArgc](#)
- char * [saArgv](#) [RF_CMD_PARAM_NUM]
- uint32_t [u32Type](#)

5.78.1 Field Documentation

5.78.1.1 iArgc

int iArgc

5.78.1.2 saArgv

char* saArgv [RF_CMD_PARAM_NUM]

5.78.1.3 u32Type

uint32_t u32Type

5.79 T_RfEvt Struct Reference

```
#include <controller_wifi_patch.h>
```

Data Fields

- void * [pParam](#)
- uint16_t [u16RfMode](#)
- uint16_t [u16RxCnt](#)
- uint16_t [u16RxCrcOkCnt](#)
- uint32_t [u32Freq](#)
- uint32_t [u32Mode](#)
- uint32_t [u32RfChannel](#)
- uint32_t [u32Type](#)
- uint8_t [u8Freq](#)
- uint8_t [u8IpcEnable](#)
- uint8_t [u8Len](#)
- uint8_t [u8Pkt](#)
- uint8_t [u8Reserved](#)
- uint8_t [u8Status](#)
- uint8_t [u8Unicast](#)

5.79.1 Field Documentation

5.79.1.1 pParam

`void* pParam`

5.79.1.2 u16RfMode

`uint16_t u16RfMode`

5.79.1.3 u16RxCnt

`uint16_t u16RxCnt`

5.79.1.4 u16RxCrcOkCnt

`uint16_t u16RxCrcOkCnt`

5.79.1.5 u32Freq

```
uint32_t u32Freq
```

5.79.1.6 u32Mode

```
uint32_t u32Mode
```

5.79.1.7 u32RfChannel

```
uint32_t u32RfChannel
```

5.79.1.8 u32Type

```
uint32_t u32Type
```

5.79.1.9 u8Freq

```
uint8_t u8Freq
```

5.79.1.10 u8IpcEnable

```
uint8_t u8IpcEnable
```

5.79.1.11 u8Len

```
uint8_t u8Len
```

5.79.1.12 u8Pkt

```
uint8_t u8Pkt
```

5.79.1.13 u8Reserved

```
uint8_t u8Reserved
```

5.79.1.14 u8Status

```
uint8_t u8Status
```

5.79.1.15 u8Unicast

```
uint8_t u8Unicast
```

5.80 wifi_active_scan_time_t Struct Reference

Range of active scan times per channel.

```
#include <wifi_types.h>
```

Data Fields

- uint32_t [max](#)
- uint32_t [min](#)

5.80.1 Detailed Description

Range of active scan times per channel.

5.80.2 Field Documentation

5.80.2.1 max

```
uint32_t max
```

maximum active scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

5.80.2.2 min

`uint32_t min`

minimum active scan time per channel, units: millisecond

5.81 wifi_ap_config_t Struct Reference

This structure is the Wi-Fi configuration for initialization for Soft-AP mode.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_auth_mode_t auth_mode](#)
- `uint16_t` [beacon_interval](#)
- `uint8_t` [channel](#)
- [wifi_cipher_type_t](#) [encrypt_type](#)
- `uint8_t` [max_connection](#)
- `uint8_t` [password](#) [WIFI_LENGTH_PASSPHRASE]
- `uint8_t` [password_length](#)
- `uint8_t` [ssid](#) [WIFI_MAX_LENGTH_OF_SSID]
- `uint8_t` [ssid_hidden](#)
- `uint8_t` [ssid_length](#)

5.81.1 Detailed Description

This structure is the Wi-Fi configuration for initialization for Soft-AP mode.

5.81.2 Field Documentation

5.81.2.1 auth_mode

`wifi_auth_mode_t` [auth_mode](#)

The authentication mode.

5.81.2.2 beacon_interval

`uint16_t` [beacon_interval](#)

Beacon interval, 100 ~ 60000 ms, default 100 ms

5.81.2.3 channel

uint8_t channel

The channel of Soft-AP.

5.81.2.4 encrypt_type

wifi_cipher_type_t encrypt_type

The encryption mode.

5.81.2.5 max_connection

uint8_t max_connection

Max number of stations allowed to connect in, default 4, max 4

5.81.2.6 password

uint8_t password[WIFI_LENGTH_PASSPHRASE]

The password of the Soft-AP.

5.81.2.7 password_length

uint8_t password_length

The length of the password.

5.81.2.8 ssid

uint8_t ssid[WIFI_MAX_LENGTH_OF_SSID]

The SSID of the Soft-AP.

5.81.2.9 ssid_hidden

uint8_t ssid_hidden

Broadcast SSID or not, default 0, broadcast the SSID

5.81.2.10 ssid_length

uint8_t ssid_length

The length of the SSID.

5.82 wifi_auto_connect_info_t Struct Reference

WiFi auto connect info parameters.

```
#include <wifi_types.h>
```

Data Fields

- uint8_t [ap_channel](#)
- uint16_t [beacon_interval](#)
- uint8_t [bssid](#) [[WIFI_MAC_ADDRESS_LENGTH](#)]
- uint16_t [capabilities](#)
- uint8_t [dtim_prod](#)
- uint8_t [fast_connect](#)
- bool [free_ocpy](#)
- int8_t [hid_ssid](#) [[WIFI_MAX_LENGTH_OF_SSID](#)]
- unsigned long [latest_beacon_rx_time](#)
- int8_t [passphrase](#) [[WIFI_LENGTH_PASSPHRASE](#)]
- uint8_t [psk](#) [32]
- uint8_t [rsn_ie](#) [100]
- int8_t [rsni](#)
- int8_t [ssid](#) [[WIFI_MAX_LENGTH_OF_SSID](#)]
- uint8_t [supported_rates](#) [[WIFI_MAX_SUPPORTED_RATES](#)]
- wpa_ie_data_t [wpa_data](#)
- uint8_t [wpa_ie](#) [100]

5.82.1 Detailed Description

WiFi auto connect info parameters.

5.82.2 Field Documentation

5.82.2.1 ap_channel

```
uint8_t ap_channel
```

5.82.2.2 beacon_interval

```
uint16_t beacon_interval
```


5.82.2.3 bssid

uint8_t bssid[WIFI_MAC_ADDRESS_LENGTH]

5.82.2.4 capabilities

uint16_t capabilities

5.82.2.5 dtim_prod

uint8_t dtim_prod

5.82.2.6 fast_connect

uint8_t fast_connect

5.82.2.7 free_ocpy

bool free_ocpy

5.82.2.8 hid_ssid

int8_t hid_ssid[WIFI_MAX_LENGTH_OF_SSID]

5.82.2.9 latest_beacon_rx_time

unsigned long latest_beacon_rx_time

5.82.2.10 passphrase

int8_t passphrase[WIFI_LENGTH_PASSPHRASE]

5.82.2.11 psk

```
uint8_t psk[32]
```

5.82.2.12 rsn_ie

```
uint8_t rsn_ie[100]
```

5.82.2.13 rssi

```
int8_t rssi
```

5.82.2.14 ssid

```
int8_t ssid[WIFI_MAX_LENGTH_OF_SSID]
```

5.82.2.15 supported_rates

```
uint8_t supported_rates[WIFI_MAX_SUPPORTED_RATES]
```

5.82.2.16 wpa_data

```
wpa_ie_data_t wpa_data
```

5.82.2.17 wpa_ie

```
uint8_t wpa_ie[100]
```

5.83 wifi_config_t Union Reference

Wi-Fi configuration for initialization.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_ap_config_t ap_config](#)
- [wifi_sta_config_t sta_config](#)

5.83.1 Detailed Description

Wi-Fi configuration for initialization.

5.83.2 Field Documentation

5.83.2.1 ap_config

[wifi_ap_config_t](#) ap_config

The configurations for certain AP. It should be set when the OPMODE is #WIFI_MODE_AP_ONLY .

5.83.2.2 sta_config

[wifi_sta_config_t](#) sta_config

The configurations for the STA. It should be set when the OPMODE is #WIFI_MODE_STA_ONLY.

5.84 wifi_event_info_t Union Reference

[wifi_event_info_t](#)

```
#include <wifi_event.h>
```

Data Fields

- [wifi_event_sta_connected_t connected](#)
- [wifi_event_sta_disconnected_t disconnected](#)
- [wifi_event_sta_got_ip_t got_ip](#)
- [wifi_event_sta_scan_done_t scan_done](#)

5.84.1 Detailed Description

[wifi_event_info_t](#)

5.84.2 Field Documentation

5.84.2.1 connected

[wifi_event_sta_connected_t](#) connected

station connected to AP

5.84.2.2 disconnected

[wifi_event_sta_disconnected_t](#) disconnected

station disconnected to AP

5.84.2.3 got_ip

[wifi_event_sta_got_ip_t](#) got_ip

station got IP, first time got IP or when IP is changed

5.84.2.4 scan_done

[wifi_event_sta_scan_done_t](#) scan_done

station scan (APs) done

5.85 [wifi_event_sta_connected_t](#) Struct Reference

[wifi_event_sta_connected_t](#)

```
#include <wifi_event.h>
```

Data Fields

- [wifi_auth_mode_t](#) authmode
- uint8_t bssid [6]
- uint8_t channel
- uint8_t ssid [32]
- uint8_t ssid_len

5.85.1 Detailed Description

[wifi_event_sta_connected_t](#)

5.85.2 Field Documentation

5.85.2.1 authmode

`wifi_auth_mode_t` authmode

5.85.2.2 bssid

`uint8_t` bssid[6]

BSSID of connected AP

5.85.2.3 channel

`uint8_t` channel

channel of connected AP

5.85.2.4 ssid

`uint8_t` ssid[32]

SSID of connected AP

5.85.2.5 ssid_len

`uint8_t` ssid_len

SSID length of connected AP

5.86 wifi_event_sta_disconnected_t Struct Reference

`wifi_event_sta_disconnected_t`

```
#include <wifi_event.h>
```

Data Fields

- `uint8_t` [bssid](#) [6]
- `uint8_t` [reason](#)
- `uint8_t` [ssid](#) [32]
- `uint8_t` [ssid_len](#)

5.86.1 Detailed Description

[wifi_event_sta_disconnected_t](#)

5.86.2 Field Documentation

5.86.2.1 bssid

```
uint8_t bssid[6]
```

BSSID of disconnected AP

5.86.2.2 reason

```
uint8_t reason
```

reason of disconnection

5.86.2.3 ssid

```
uint8_t ssid[32]
```

SSID of disconnected AP

5.86.2.4 ssid_len

```
uint8_t ssid_len
```

SSID length of disconnected AP

5.87 wifi_event_sta_got_ip_t Struct Reference

[wifi_event_sta_got_ip_t](#)

```
#include <wifi_event.h>
```

Data Fields

- bool [ip_changed](#)

5.87.1 Detailed Description

[wifi_event_sta_got_ip_t](#)

5.87.2 Field Documentation

5.87.2.1 ip_changed

bool ip_changed

5.88 wifi_event_sta_scan_done_t Struct Reference

[wifi_event_sta_scan_done_t](#)

```
#include <wifi_event.h>
```

Data Fields

- uint8_t [number](#)
- uint8_t [scan_id](#)
- uint32_t [status](#)

5.88.1 Detailed Description

[wifi_event_sta_scan_done_t](#)

5.88.2 Field Documentation

5.88.2.1 number

uint8_t number

The number of devices scanned

5.88.2.2 scan_id

uint8_t scan_id

scan id

5.88.2.3 status

`uint32_t status`

status of scanning APs

5.89 wifi_fast_scan_threshold_t Struct Reference

Structure describing parameters for a Wi-Fi fast scan.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_auth_mode_t](#) authmode
- `int8_t` rssi

5.89.1 Detailed Description

Structure describing parameters for a Wi-Fi fast scan.

5.89.2 Field Documentation

5.89.2.1 authmode

`wifi_auth_mode_t` authmode

The weakest authmode to accept in the fast scan mode

5.89.2.2 rssi

`int8_t` rssi

The minimum rssi to accept in the fast scan mode

5.90 wifi_init_config_t Struct Reference

WiFi stack configuration parameters.

```
#include <wifi_types.h>
```


Data Fields

- [wifi_event_notify_cb_t](#) event_handler
- int magic

5.90.1 Detailed Description

WiFi stack configuration parameters.

5.90.2 Field Documentation

5.90.2.1 event_handler

[wifi_event_notify_cb_t](#) event_handler

WiFi event handler

5.90.2.2 magic

int magic

WiFi init magic number, it should be the last field

5.91 wifi_scan_config_t Struct Reference

Parameters for an SSID scan.

```
#include <wifi_types.h>
```

Data Fields

- uint8_t * bssid
- uint8_t channel
- [wifi_scan_time_t](#) scan_time
- [wifi_scan_type_t](#) scan_type
- bool show_hidden
- uint8_t * ssid

5.91.1 Detailed Description

Parameters for an SSID scan.

5.91.2 Field Documentation

5.91.2.1 bssid

```
uint8_t* bssid
```

MAC address of AP

5.91.2.2 channel

```
uint8_t channel
```

channel, scan the specific channel

5.91.2.3 scan_time

```
wifi_scan_time_t scan_time
```

scan time per channel

5.91.2.4 scan_type

```
wifi_scan_type_t scan_type
```

scan type, active or passive

5.91.2.5 show_hidden

```
bool show_hidden
```

enable to scan AP whose SSID is hidden

5.91.2.6 ssid

```
uint8_t* ssid
```

SSID of AP

5.92 wifi_scan_info_t Struct Reference

This structure defines the information of scanned APs.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_auth_mode_t](#) auth_mode
- [uint16_t](#) beacon_interval
- [uint8_t](#) bssid [[WIFI_MAC_ADDRESS_LENGTH](#)]
- [uint16_t](#) capability_info
- [uint8_t](#) channel
- [wifi_cipher_type_t](#) group_cipher
- [wifi_cipher_type_t](#) pairwise_cipher
- [int](#) rssi
- [uint8_t](#) ssid [[WIFI_MAX_LENGTH_OF_SSID](#)]
- [uint8_t](#) ssid_length

5.92.1 Detailed Description

This structure defines the information of scanned APs.

5.92.2 Field Documentation

5.92.2.1 auth_mode

[wifi_auth_mode_t](#) auth_mode

Please refer to the definition of [wifi_auth_mode_t](#).

5.92.2.2 beacon_interval

[uint16_t](#) beacon_interval

Indicates the beacon interval.

5.92.2.3 bssid

[uint8_t](#) bssid [[WIFI_MAC_ADDRESS_LENGTH](#)]

AP's MAC address.

5.92.2.4 capability_info

[uint16_t](#) capability_info

The Capability Information field contains a number of subfields that are used to indicate requested or advertised optional capabilities.

5.92.2.5 channel

```
uint8_t channel
```

The channel used.

5.92.2.6 group_cipher

```
wifi_cipher_type_t group_cipher
```

group cipher of AP

5.92.2.7 pairwise_cipher

```
wifi_cipher_type_t pairwise_cipher
```

pairwise cipher of AP, Please refer to the definition of #wifi_encrypt_type_t.

5.92.2.8 rssi

```
int rssi
```

Records the RSSI value when probe response is received.

5.92.2.9 ssid

```
uint8_t ssid[WIFI_MAX_LENGTH_OF_SSID]
```

Stores the predefined SSID.

5.92.2.10 ssid_length

```
uint8_t ssid_length
```

Length of the SSID.

5.93 wifi_scan_list_t Struct Reference

This structure defines the list of scanned APs with their corresponding information.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_scan_info_t ap_record](#) [WIFI_MAX_SCAN_AP_NUM]
- int [num](#)

5.93.1 Detailed Description

This structure defines the list of scanned APs with their corresponding information.

5.93.2 Field Documentation

5.93.2.1 ap_record

```
wifi_scan_info_t ap_record[WIFI_MAX_SCAN_AP_NUM]
```

The information about an AP obtained through the scan result is stored

5.93.2.2 num

```
int num
```

number of AP in the list

5.94 wifi_scan_time_t Union Reference

Aggregate of active & passive scan time per channel.

```
#include <wifi_types.h>
```

Data Fields

- [wifi_active_scan_time_t](#) active
- [uint32_t](#) passive

5.94.1 Detailed Description

Aggregate of active & passive scan time per channel.

5.94.2 Field Documentation

5.94.2.1 active

```
wifi_active_scan_time_t active
```

active scan time per channel, units: millisecond.

5.94.2.2 passive

uint32_t passive

passive scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

5.95 wifi_sta_config_t Struct Reference

This structure is the Wi-Fi configuration for initialization for STA mode.

```
#include <wifi_types.h>
```

Data Fields

- uint8_t [bssid](#) [[WIFI_MAC_ADDRESS_LENGTH](#)]
- uint8_t [bssid_present](#)
- uint8_t [password](#) [[WIFI_LENGTH_PASSPHRASE](#)]
- uint8_t [password_length](#)
- [wifi_scan_method_t](#) [scan_method](#)
- [wifi_sort_method_t](#) [sort_method](#)
- uint8_t [ssid](#) [[WIFI_MAX_LENGTH_OF_SSID](#)]
- uint8_t [ssid_length](#)
- [wifi_fast_scan_threshold_t](#) [threshold](#)

5.95.1 Detailed Description

This structure is the Wi-Fi configuration for initialization for STA mode.

5.95.2 Field Documentation

5.95.2.1 bssid

uint8_t [bssid](#) [[WIFI_MAC_ADDRESS_LENGTH](#)]

The MAC address of the target AP.

5.95.2.2 bssid_present

uint8_t [bssid_present](#)

The BSSID is present if it is set to 1. Otherwise, it is set to 0.

5.95.2.3 password

```
uint8_t password[WIFI_LENGTH_PASSPHRASE]
```

The password of the target AP.

5.95.2.4 password_length

```
uint8_t password_length
```

The length of the password. If the length is 64, the password is regarded as PMK.

5.95.2.5 scan_method

```
wifi_scan_method_t scan_method
```

do all channel scan or fast scan

5.95.2.6 sort_method

```
wifi_sort_method_t sort_method
```

sort the connect AP in the list by rssi or security mode

5.95.2.7 ssid

```
uint8_t ssid[WIFI_MAX_LENGTH_OF_SSID]
```

The SSID of the target AP.

5.95.2.8 ssid_length

```
uint8_t ssid_length
```

The length of the SSID.

5.95.2.9 threshold

```
wifi_fast_scan_threshold_t threshold
```

When scan_method is set to WIFI_FAST_SCAN, only APs which have an auth mode that is more secure than the selected auth mode and a signal stronger than the minimum RSSI will be used.

Index

- action
 - LE_SMP_MSG_PAIRING_ACTION_IND_T, [194](#)
- active
 - wifi_scan_time_t, [223](#)
- addr
 - LE_BT_ADDR_T, [137](#)
 - LE_CM_MSG_ADVERTISE_REPORT_IND_↔
T, [140](#)
- addr_type
 - LE_CM_MSG_ADVERTISE_REPORT_IND_↔
T, [140](#)
- ap_channel
 - auto_conn_info_t, [131](#)
 - mw_wifi_auto_connect_ap_info_t, [201](#)
 - wifi_auto_connect_info_t, [210](#)
- ap_config
 - wifi_config_t, [213](#)
- ap_record
 - wifi_scan_list_t, [223](#)
- att_err
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IND_↔
T, [166](#)
 - LE_GATT_MSG_EXECUTE_WRITE_RELIABL_↔
E_CFM_T, [170](#)
 - LE_GATT_MSG_FIND_ALL_CHAR_DESC_CF_↔
M_T, [171](#)
 - LE_GATT_MSG_FIND_ALL_PRIMARY_SERVI_↔
CE_CFM_T, [172](#)
 - LE_GATT_MSG_FIND_CHARACTERISTIC_CF_↔
M_T, [173](#)
 - LE_GATT_MSG_FIND_INCLUDED_SERVICE_↔
CFM_T, [174](#)
 - LE_GATT_MSG_FIND_PRIMARY_SERVICE_B_↔
Y_UUID_CFM_T, [175](#)
 - LE_GATT_MSG_PREPARE_WRITE_RELIABL_↔
E_CFM_T, [181](#)
 - LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_↔
_CFM_T, [182](#)
 - LE_GATT_MSG_READ_CHARACTERISTIC_V_↔
ALUE_CFM_T, [183](#)
 - LE_GATT_MSG_READ_LONG_CHAR_VAL_C_↔
FM_T, [184](#)
 - LE_GATT_MSG_READ_MULTIPLE_CHAR_VA_↔
L_CFM_T, [185](#)
 - LE_GATT_MSG_WRITE_CHAR_VAL_RELIAB_↔
LE_CFM_T, [188](#)
 - LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_↔
_T, [189](#)
 - LE_GATT_MSG_WRITE_LONG_CHAR_VALU_↔
E_CFM_T, [190](#)
- att_op
 - LE_GATT_MSG_OPERATION_TIMEOUT_T, [180](#)
- auth_mode
 - wifi_ap_config_t, [208](#)
 - wifi_scan_info_t, [221](#)
- authenticated
 - LE_SMP_MSG_PAIRING_COMPLETE_IND_T,
[195](#)
- authmode
 - wifi_event_sta_connected_t, [215](#)
 - wifi_fast_scan_threshold_t, [218](#)
- auto_conn_info_t, [131](#)
 - ap_channel, [131](#)
 - beacon_interval, [131](#)
 - bssid, [132](#)
 - capabilities, [132](#)
 - dtim_prod, [132](#)
 - fast_connect, [132](#)
 - free_ocpy, [132](#)
 - hid_ssid, [132](#)
 - latest_beacon_rx_time, [132](#)
 - passphrase, [132](#)
 - psk, [133](#)
 - rsn_ie, [133](#)
 - rsssi, [133](#)
 - ssid, [133](#)
 - supported_rates, [133](#)
 - wpa_data, [133](#)
 - wpa_ie, [133](#)
- auto_connect_cfg_t, [134](#)
 - flag, [134](#)
 - front, [134](#)
 - max_save_num, [134](#)
 - pFCInfo, [134](#)
 - rear, [134](#)
 - retryCount, [135](#)
 - targetIdx, [135](#)
 - uFCAPNum, [135](#)
- BLE ALL APIs, [9](#)
- BLE CM APIs, [10](#)
 - LE_CM_MSG_ADD_TO_RESOLVING_LIST_C_↔
FM_T, [11](#)
 - LE_CM_MSG_ADD_TO_WHITE_LIST_CFM_T,
[11](#)
 - LE_CM_MSG_CANCEL_CONNECTION_CFM_T,
[11](#)
 - LE_CM_MSG_CLEAR_RESOLVING_LIST_CF_↔
M_T, [12](#)

- LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T, [12](#)
- LE_CM_MSG_CREATE_CONNECTION_CFM_T, [12](#)
- LE_CM_MSG_ENTER_ADVERTISING_CFM_T, [12](#)
- LE_CM_MSG_ENTER_SCANNING_CFM_T, [12](#)
- LE_CM_MSG_EXIT_ADVERTISING_CFM_T, [12](#)
- LE_CM_MSG_EXIT_SCANNING_CFM_T, [12](#)
- LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T, [12](#)
- LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T, [13](#)
- LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T, [13](#)
- LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T, [13](#)
- LE_CM_MSG_SET_CHANNEL_MAP_CFM_T, [13](#)
- LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T, [13](#)
- LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T, [13](#)
- LE_CM_MSG_SET_SCAN_PARAMS_CFM_T, [13](#)
- LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T, [13](#)
- LeCmlInit, [15](#)
- BLE GAP APIs, [16](#)
 - GAP_ADTYPE_128BIT_COMPLETE, [18](#)
 - GAP_ADTYPE_128BIT_MORE, [18](#)
 - GAP_ADTYPE_16BIT_COMPLETE, [18](#)
 - GAP_ADTYPE_16BIT_MORE, [18](#)
 - GAP_ADTYPE_32BIT_COMPLETE, [19](#)
 - GAP_ADTYPE_32BIT_MORE, [19](#)
 - GAP_ADTYPE_3D_INFO_DATA, [19](#)
 - GAP_ADTYPE_ADV_INTERVAL, [19](#)
 - GAP_ADTYPE_APPEARANCE, [19](#)
 - GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED, [19](#)
 - GAP_ADTYPE_FLAGS_GENERAL, [19](#)
 - GAP_ADTYPE_FLAGS_LIMITED, [20](#)
 - GAP_ADTYPE_FLAGS, [19](#)
 - GAP_ADTYPE_LE_BD_ADDR, [20](#)
 - GAP_ADTYPE_LE_ROLE, [20](#)
 - GAP_ADTYPE_LOCAL_NAME_COMPLETE, [20](#)
 - GAP_ADTYPE_LOCAL_NAME_SHORT, [20](#)
 - GAP_ADTYPE_MANUFACTURER_SPECIFIC, [20](#)
 - GAP_ADTYPE_OOB_CLASS_OF_DEVICE, [20](#)
 - GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC, [20](#)
 - GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDOM, [21](#)
 - GAP_ADTYPE_POWER_LEVEL, [21](#)
 - GAP_ADTYPE_PUBLIC_TARGET_ADDR, [21](#)
 - GAP_ADTYPE_RANDOM_TARGET_ADDR, [21](#)
 - GAP_ADTYPE_SERVICE_DATA_128BIT, [21](#)
 - GAP_ADTYPE_SERVICE_DATA_32BIT, [21](#)
 - GAP_ADTYPE_SERVICE_DATA, [21](#)
 - GAP_ADTYPE_SERVICES_LIST_128BIT, [21](#)
 - GAP_ADTYPE_SERVICES_LIST_16BIT, [22](#)
 - GAP_ADTYPE_SIGNED_DATA, [22](#)
 - GAP_ADTYPE_SIMPLE_PAIRING_HASHC, [256](#)
 - GAP_ADTYPE_SIMPLE_PAIRING_RANDOM, [256](#)
 - GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANDOM, [22](#)
 - GAP_ADTYPE_SM_OOB_FLAG, [22](#)
 - GAP_ADTYPE_SM_TK, [22](#)
 - GAP_PUBLIC_ADDR, [22](#)
 - GAP_RANDOM_ADDR_NRPA, [23](#)
 - GAP_RANDOM_ADDR_RPA, [23](#)
 - GAP_RANDOM_ADDR_STATIC, [23](#)
 - GAP_SCAN_TYPE_ACTIVE, [23](#)
 - GAP_SCAN_TYPE_PASSIVE, [23](#)
 - GAP_TX_PWR_CURR_VAL, [23](#)
 - GAP_TX_PWR_MAX_VAL, [23](#)
 - GAPBOND_IO_CAP_DISPLAY_ONLY, [23](#)
 - GAPBOND_IO_CAP_DISPLAY_YES_NO, [24](#)
 - GAPBOND_IO_CAP_KEYBOARD_DISPLAY, [24](#)
 - GAPBOND_IO_CAP_KEYBOARD_ONLY, [24](#)
 - GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT, [24](#)
 - GAPBOND_PAIRING_MODE_INITIATE, [24](#)
 - GAPBOND_PAIRING_MODE_NO_PAIRING, [24](#)
 - GAPBOND_PAIRING_MODE_WAIT_FOR_REQ, [24](#)
 - LE_GAP_ADV_MAX_SIZE, [24](#)
 - LeGapAddToResolvingList, [25](#)
 - LeGapAddToWhiteList, [25](#)
 - LeGapAdvertisingEnable, [25](#)
 - LeGapCentralConnectReq, [26](#)
 - LeGapCentralSetDataChannel, [26](#)
 - LeGapClearResolvingList, [27](#)
 - LeGapClearWhiteList, [27](#)
 - LeGapConnParaRequestRsp, [27](#)
 - LeGapConnUpdateRequest, [28](#)
 - LeGapConnUpdateResponse, [28](#)
 - LeGapConnectCancelReq, [27](#)
 - LeGapDisconnectReq, [29](#)
 - LeGapGenRandAddr, [29](#)
 - LeGapGetBtAddr, [29](#)
 - LeGapReadAdvChannelTxPower, [29](#)
 - LeGapReadChannelMap, [30](#)
 - LeGapReadResolvingListSize, [30](#)
 - LeGapReadRssi, [30](#)
 - LeGapReadTxPower, [31](#)
 - LeGapReadWhiteListSize, [31](#)
 - LeGapRemoveFromWhiteList, [31](#)
 - LeGapScanningReq, [32](#)
 - LeGapSetAdvData, [32](#)
 - LeGapSetAdvParameter, [33](#)
 - LeGapSetConnParameter, [33](#)
 - LeGapSetDataChannelPduLen, [33](#)
 - LeGapSetRandAddr, [34](#)
 - LeGapSetRpaTimeout, [34](#)
 - LeGapSetStaticAddr, [35](#)
 - LeSetScanParameter, [35](#)
 - LeSetScanRspData, [35](#)

- BLE GATT APIs, 37
 - CHARAggregateDescriptor, 41
 - CHARClientConfigDescriptor, 41
 - CHAR_DECL_UUID16_ATTR_VAL, 42
 - CHAR_EXT_PROP_DESCRIPTOR, 42
 - CHAR_PRESENT_FORMAT_DESCRIPTOR, 42
 - CHAR_SERVER_CONFIG_DESCRIPTOR, 42
 - CHAR_USER_DESC_DESCRIPTOR, 42
 - CHARACTERISTIC_DECL_UUID128, 42
 - CHARACTERISTIC_DECL_UUID16, 43
 - CHARACTERISTIC_UUID128, 43
 - CHARACTERISTIC_UUID16, 43
 - GATT_CHAR_AGG_FORMAT_UUID, 43
 - GATT_CHAR_EXT_PROPS_UUID, 43
 - GATT_CHAR_FORMAT_UUID, 43
 - GATT_CHAR_USER_DESC_UUID, 44
 - GATT_CHARACTERISTIC_UUID, 44
 - GATT_CLIENT_CHAR_CFG_UUID, 44
 - GATT_EXT_REPORT_REF_UUID, 44
 - GATT_INCLUDE_UUID, 44
 - GATT_PRIMARY_SERVICE_UUID, 44
 - GATT_REPORT_REF_UUID, 44
 - GATT_SECONDARY_SERVICE_UUID, 44
 - GATT_SERV_CHAR_CFG_UUID, 45
 - GATT_VALID_RANGE_UUID, 45
 - gcCharAggregateUuid, 68
 - gcCharExtPropUuid, 68
 - gcCharFormatUuid, 69
 - gcCharUserDescUuid, 69
 - gcCharacteristicUuid, 68
 - gcClientCharConfigUuid, 69
 - gcExtReportRefUuid, 69
 - gcIncludeUuid, 69
 - gcPrimaryServiceUuid, 69
 - gcReportRefUuid, 69
 - gcSecondaryServiceUuid, 69
 - gcServerCharConfigUuid, 70
 - gcValidRangeUuid, 70
 - INCLUDE_DECL_UUID128, 45
 - INCLUDE_DECL_UUID128_ATTR_VAL, 45
 - INCLUDE_DECL_UUID16_ATTR_VAL, 45
 - INCLUDE_DECL_UUID16, 45
 - LE_ATT_UUID_SIZE, 45
 - LE_GATT_CHAR_PROP_AUTH, 46
 - LE_GATT_CHAR_PROP_BCAST, 46
 - LE_GATT_CHAR_PROP_EXT_PROP, 46
 - LE_GATT_CHAR_PROP_IND, 46
 - LE_GATT_CHAR_PROP_NTF, 46
 - LE_GATT_CHAR_PROP_RD, 46
 - LE_GATT_CHAR_PROP_WR_NO_RESP, 47
 - LE_GATT_CHAR_PROP_WR, 46
 - LE_GATT_CLIENT_CFG_INDICATION, 47
 - LE_GATT_CLIENT_CFG_NOTIFICATION, 47
 - LE_GATT_EXT_PROP_RELIABLE_WR, 47
 - LE_GATT_EXT_PROP_WR_AUX, 47
 - LE_GATT_FLAG_PREPARE_WRITE, 47
 - LE_GATT_FLAG_WRITE_CMD, 47
 - LE_GATT_FLAG_WRITE_REQ, 47
 - LE_GATT_PERM_AUTH_READABLE, 48
 - LE_GATT_PERM_AUTH_WRITABLE, 48
 - LE_GATT_PERM_NONE, 48
 - LE_GATT_PERM_READ, 48
 - LE_GATT_PERM_RELIABLE_WRITE, 48
 - LE_GATT_PERM_WRITE_CMD, 48
 - LE_GATT_PERM_WRITE_REQ, 48
 - LE_GATT_PERMIT_AUTHEN_READ, 48
 - LE_GATT_PERMIT_AUTHEN_WRITE, 49
 - LE_GATT_PERMIT_AUTHOR_READ, 49
 - LE_GATT_PERMIT_AUTHOR_WRITE, 49
 - LE_GATT_PERMIT_ENCRYPT_READ, 49
 - LE_GATT_PERMIT_ENCRYPT_WRITE, 49
 - LE_GATT_PERMIT_READABLE, 49
 - LE_GATT_PERMIT_READ, 49
 - LE_GATT_PERMIT_SC_AUTHEN_READ, 49
 - LE_GATT_PERMIT_SC_AUTHEN_WRITE, 50
 - LE_GATT_PERMIT_WRITABLE, 50
 - LE_GATT_PERMIT_WRITE, 50
 - LeGattAccessReadRsp, 52
 - LeGattAccessWriteRsp, 52
 - LeGattChangeAttrVal, 53
 - LeGattCharValConfirmation, 53
 - LeGattCharValIndicate, 54
 - LeGattCharValNotify, 54
 - LeGattExchangeMtuReq, 55
 - LeGattExchangeMtuRsp, 55
 - LeGattExecuteWriteCharValReliable, 55
 - LeGattFindAllCharDescriptor, 56
 - LeGattFindAllCharacteristic, 56
 - LeGattFindAllPrimaryService, 57
 - LeGattFindCharacteristicByUuid, 57
 - LeGattFindIncludedService, 58
 - LeGattFindPrimaryServiceByUuid, 58
 - LeGattGetAttrHandle, 58
 - LeGattGetAttrVal, 59
 - LeGattGetAttrValLen, 59
 - LeGattGetAttrValMaxLen, 61
 - LeGattInit, 61
 - LeGattModifyAttrVal, 62
 - LeGattPrepareWriteCharValReliable, 62
 - LeGattReadCharValByUuid, 63
 - LeGattReadCharValue, 63
 - LeGattReadLongCharVal, 64
 - LeGattReadMultipleCharVal, 64
 - LeGattRegisterIncludeService, 64
 - LeGattRegisterService, 65
 - LeGattSignedWriteNoRsp, 65
 - LeGattStopCurrentProcedure, 66
 - LeGattWriteCharVal, 66
 - LeGattWriteCharValReliable, 67
 - LeGattWriteLongCharVal, 67
 - LeGattWriteNoRsp, 68
 - PRIMARY_SERVICE_DECL_UUID128, 50
 - PRIMARY_SERVICE_DECL_UUID16, 50
 - SECONDARY_SERVICE_DECL_UUID128, 50
 - SECONDARY_SERVICE_DECL_UUID16, 50
- BLE MSG APIs, 71

- LE_ATT_MSG_BASE, 72
- LE_CM_MSG_BASE, 72
- LE_GATT_MSG_BASE, 72
- LE_HCI_MSG_BASE, 73
- LE_L2CAP_MSG_BASE, 73
- LE_SMP_MSG_BASE, 73
- LE_SYS_MSG_BASE, 73
- LeCancelAllMessage, 76
- LeCancelAllSubMessage, 77
- LeCancelFirstMessage, 77
- LeCancelFirstSubMessage, 77
- LeGetSubMsgId, 78
- LeHostCreateTask, 78
- LeHostMessageLoop, 79
- LeSendMessage, 79
- LeSendMessageAfter, 79
- LeSendMessageUnlock, 80
- LeSendSubMessage, 80
- LeSendSubMessageAfter, 81
- LeSendSubMessageUnlock, 81
- MESSAGE_ALLOCATE, 73
- MESSAGE_BULID, 73
- MESSAGE_DATA_BULID, 73
- MESSAGE_OFFSET, 74
- MESSAGEID, 74
- MESSAGE, 74
- MSGLOCK, 75
- MSGSUBID, 75
- MSGTIMER, 75
- MsgData, 75
- MsgLock, 75
- T_HOUR, 74
- T_MIN, 74
- T_SEC, 74
- TASKHANDLER, 75
- TASKPACK, 76
- TASK, 75
- Task, 75
- BLE SMP APIs, 83
 - LE_MAX_BOND_COUNT, 84
 - LE_SM_IO_CAP_DISP_ONLY, 84
 - LE_SM_IO_CAP_DISP_YES_NO, 84
 - LE_SM_IO_CAP_KEYBOARD_DISP, 84
 - LE_SM_IO_CAP_KEYBOARD_ONLY, 85
 - LE_SM_IO_CAP_NO_IO, 85
 - LE_SM_PAIR_MITM_NO, 85
 - LE_SM_PAIR_MITM_YES, 85
 - LE_SM_PAIR_OOB_NO, 85
 - LE_SM_PAIR_OOB_YES, 85
 - LE_SM_PAIR_SC_NO, 85
 - LE_SM_PAIR_SC_YES, 85
 - LeSmpInit, 87
 - LeSmpOobAuthDataRsp, 87
 - LeSmpOobPresent, 87
 - LeSmpPasskeyInput, 88
 - LeSmpScOobComputeConfirmVal, 88
 - LeSmpScOobDataRsp, 88
 - LeSmpSecurityReq, 89
 - LeSmpSecurityRsp, 89
 - LeSmpSetDefaultConfig, 90
 - LeSmpUserConfirmRsp, 90
- bd_addr
 - LE_CM_MSG_READ_BD_ADDR_CFM_T, 149
- beacon_interval
 - auto_conn_info_t, 131
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_ap_config_t, 208
 - wifi_auto_connect_info_t, 210
 - wifi_scan_info_t, 221
- bondable
 - LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T, 198
- bonded
 - LE_SMP_MSG_PAIRING_COMPLETE_IND_T, 195
- bssid
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 210
 - wifi_event_sta_connected_t, 215
 - wifi_event_sta_disconnected_t, 216
 - wifi_scan_config_t, 220
 - wifi_scan_info_t, 221
 - wifi_sta_config_t, 224
- bssid_present
 - wifi_sta_config_t, 224
- CHARAggregateDescriptor
 - BLE GATT APIs, 41
- CHARClientConfigDescriptor
 - BLE GATT APIs, 41
- CHARDeclUUID16AttrVal
 - BLE GATT APIs, 42
- CHARExtPropDescriptor
 - BLE GATT APIs, 42
- CHARPresentFormatDescriptor
 - BLE GATT APIs, 42
- CHARServerConfigDescriptor
 - BLE GATT APIs, 42
- CHARUserDescDescriptor
 - BLE GATT APIs, 42
- CHARACTERISTICDeclUUID128
 - BLE GATT APIs, 42
- CHARACTERISTICDeclUUID16
 - BLE GATT APIs, 43
- CHARACTERISTICUUID128
 - BLE GATT APIs, 43
- CHARACTERISTICUUID16
 - BLE GATT APIs, 43
- capabilities
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 211
- capability_info
 - wifi_scan_info_t, 221
- ch_map

- LE_CM_MSG_READ_CHANNEL_MAP_CFM_T, 150
- channel
 - wifi_ap_config_t, 208
 - wifi_event_sta_connected_t, 215
 - wifi_scan_config_t, 220
 - wifi_scan_info_t, 221
- channel_map
 - LE_GAP_ADVERTISING_PARAM_T, 157
- client_rx_mtu
 - LE_GATT_MSG_EXCHANGE_MTU_IND_T, 169
- confirm
 - LE_SMP_SC_OOB_DATA_T, 199
- confirm_num
 - LE_SMP_MSG_USER_CONFIRM_IND_T, 199
- conn_hdl
 - LE_CM_CONNECTION_COMPLETE_IND_T, 138
 - LE_CM_MSG_CONN_PARA_REQ_T, 141
 - LE_CM_MSG_CONN_UPDATE_COMPLETE_I↔ND_T, 142
 - LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 143
 - LE_CM_MSG_DISCONNECT_COMPLETE_IN↔D_T, 145
 - LE_CM_MSG_ENCRYPTION_CHANGE_IND_T, 145
 - LE_CM_MSG_ENCRYPTION_REFRESH_IND_T, 146
 - LE_CM_MSG_LTK_REQ_IND_T, 148
 - LE_CM_MSG_READ_CHANNEL_MAP_CFM_T, 150
 - LE_CM_MSG_READ_RSSI_CFM_T, 151
 - LE_CM_MSG_READ_TX_POWER_CFM_T, 152
 - LE_CM_MSG_SET_DATA_LENGTH_CFM_T, 153
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 154
 - LE_GATT_MSG_ACCESS_READ_IND_T, 162
 - LE_GATT_MSG_ACCESS_WRITE_IND_T, 162
 - LE_GATT_MSG_CHAR_DESCRIPTOR_INFO↔IND_T, 164
 - LE_GATT_MSG_CHARACTERISTIC_DECL_IN↔FO_IND_T, 165
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IN↔T, 166
 - LE_GATT_MSG_CONFIRMATION_CFM_T, 167
 - LE_GATT_MSG_EXCHANGE_MTU_CFM_T, 168
 - LE_GATT_MSG_EXCHANGE_MTU_IND_T, 169
 - LE_GATT_MSG_EXECUTE_WRITE_RELIABL↔E_CFM_T, 170
 - LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T, 171
 - LE_GATT_MSG_FIND_ALL_PRIMARY_SERVI↔CE_CFM_T, 172
 - LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T, 173
 - LE_GATT_MSG_FIND_INCLUDED_SERVICE↔CFM_T, 174
 - LE_GATT_MSG_FIND_PRIMARY_SERVICE_B↔Y_UUID_CFM_T, 175
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_I↔ND_T, 176
 - LE_GATT_MSG_INDICATE_IND_T, 177
 - LE_GATT_MSG_NOTIFY_CFM_T, 178
 - LE_GATT_MSG_NOTIFY_IND_T, 179
 - LE_GATT_MSG_OPERATION_TIMEOUT_T, 180
 - LE_GATT_MSG_PREPARE_WRITE_RELIABL↔E_CFM_T, 181
 - LE_GATT_MSG_READ_CHAR_VAL_BY_UUID↔CFM_T, 182
 - LE_GATT_MSG_READ_CHARACTERISTIC_V↔ALUE_CFM_T, 183
 - LE_GATT_MSG_READ_LONG_CHAR_VAL_C↔FM_T, 184
 - LE_GATT_MSG_READ_MULTIPLE_CHAR_VA↔L_CFM_T, 185
 - LE_GATT_MSG_SERVICE_INFO_IND_T, 186
 - LE_GATT_MSG_SIGNED_WRITE_CFM_T, 187
 - LE_GATT_MSG_WRITE_CHAR_VAL_RELIAB↔LE_CFM_T, 188
 - LE_GATT_MSG_WRITE_CHAR_VALUE_CFM↔T, 189
 - LE_GATT_MSG_WRITE_LONG_CHAR_VALU↔E_CFM_T, 190
 - LE_GATT_MSG_WRITE_NO_RSP_CFM_T, 191
 - LE_SMP_MSG_ENCRYPTION_CHANGE_IN↔T, 193
 - LE_SMP_MSG_ENCRYPTION_REFRESH_IN↔T, 193
 - LE_SMP_MSG_OOB_DATA_REQUEST_IND_T, 194
 - LE_SMP_MSG_PAIRING_ACTION_IND_T, 194
 - LE_SMP_MSG_PAIRING_COMPLETE_IND_T, 195
 - LE_SMP_MSG_PASSKEY_DISPLAY_IND_T, 196
 - LE_SMP_MSG_PASSKEY_INPUT_IND_T, 197
 - LE_SMP_MSG_SC_OOB_DATA_REQUEST_I↔ND_T, 197
 - LE_SMP_MSG_SLAVE_SECURITY_REQUES↔T_IND_T, 198
 - LE_SMP_MSG_USER_CONFIRM_IND_T, 199
 - LE_SYS_MSG_BUF_OVERFLOW_T, 200
- conn_interval
 - LE_CM_CONNECTION_COMPLETE_IND_T, 138
- conn_latency
 - LE_CM_CONNECTION_COMPLETE_IND_T, 138
- connected
 - wifi_event_info_t, 214
- current_rx_mtu
 - LE_GATT_MSG_EXCHANGE_MTU_CFM_T, 168
- data
 - LE_CM_MSG_ADVERTISE_REPORT_IND↔T, 140
- dev_id
 - LE_CM_CONNECTION_COMPLETE_IND_T, 138
- devid
 - LE_CM_MSG_ENCRYPTION_CHANGE_IND_T, 146

- LE_CM_MSG_ENCRYPTION_REFRESH_IND_T, 146
- LE_CM_MSG_LTK_REQ_IND_T, 148
- LE_GATT_MSG_ACCESS_READ_IND_T, 162
- LE_GATT_MSG_ACCESS_WRITE_IND_T, 163
- LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T, 164
- LE_GATT_MSG_CHARACTERISTIC_DECL_IND_T, 165
- LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T, 166
- LE_GATT_MSG_CONFIRMATION_CFM_T, 168
- LE_GATT_MSG_EXCHANGE_MTU_CFM_T, 168
- LE_GATT_MSG_EXCHANGE_MTU_IND_T, 169
- LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T, 170
- LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T, 171
- LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T, 172
- LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T, 173
- LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T, 174
- LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T, 175
- LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T, 176
- LE_GATT_MSG_INDICATE_IND_T, 177
- LE_GATT_MSG_NOTIFY_CFM_T, 178
- LE_GATT_MSG_NOTIFY_IND_T, 179
- LE_GATT_MSG_OPERATION_TIMEOUT_T, 180
- LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T, 181
- LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T, 182
- LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T, 183
- LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T, 184
- LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T, 185
- LE_GATT_MSG_SERVICE_INFO_IND_T, 186
- LE_GATT_MSG_SIGNED_WRITE_CFM_T, 187
- LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T, 188
- LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T, 189
- LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T, 190
- LE_GATT_MSG_WRITE_NO_RSP_CFM_T, 191
- direct_addr
 - LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 144
- direct_addr_type
 - LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 144
- disconnected
 - wifi_event_info_t, 214
- dtim_prod
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 211
- ediv
 - LE_CM_MSG_LTK_REQ_IND_T, 148
- enable
 - LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T, 193
- enabled
 - LE_CM_MSG_ENCRYPTION_CHANGE_IND_T, 146
 - LE_CM_MSG_ENCRYPTION_REFRESH_IND_T, 147
- encrypt_type
 - wifi_ap_config_t, 209
- end_hdl
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T, 176
 - LE_GATT_MSG_SERVICE_INFO_IND_T, 186
- endHdl
 - LE_GATT_SERVICE_T, 192
- Enumeration, 125
 - wifi_auth_mode_t, 125
 - wifi_bandwidth_t, 126
 - wifi_cipher_type_t, 126
 - wifi_event_t, 126
 - wifi_mode_t, 127
 - wifi_reason_code_t, 127
 - wifi_scan_method_t, 128
 - wifi_scan_type_t, 128
 - wifi_sort_method_t, 129
- err_hdl
 - LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T, 170
 - LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T, 185
- event
 - event_msg_t, 135
- event_handler
 - wifi_init_config_t, 219
- event_msg_t, 135
 - event, 135
 - length, 136
 - param, 136
- event_type
 - LE_CM_MSG_ADVERTISE_REPORT_IND_T, 140
- fast_connect
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 211
- filter_policy
 - LE_GAP_ADVERTISING_PARAM_T, 157
 - LE_GAP_SCAN_PARAM_T, 159
- flag

- auto_connect_cfg_t, 134
- LE_GATT_MSG_ACCESS_WRITE_IND_T, 163
- MwFimAutoConnectCFG_t, 203
- format
 - LE_GATT_ATTR_T, 160
 - LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T, 164
 - LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T, 165
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T, 176
 - LE_GATT_MSG_SERVICE_INFO_IND_T, 186
- free_ocpy
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 211
- front
 - auto_connect_cfg_t, 134
 - MwFimAutoConnectCFG_t, 203
- GAP_ADTYPE_128BIT_COMPLETE
 - BLE GAP APIs, 18
- GAP_ADTYPE_128BIT_MORE
 - BLE GAP APIs, 18
- GAP_ADTYPE_16BIT_COMPLETE
 - BLE GAP APIs, 18
- GAP_ADTYPE_16BIT_MORE
 - BLE GAP APIs, 18
- GAP_ADTYPE_32BIT_COMPLETE
 - BLE GAP APIs, 19
- GAP_ADTYPE_32BIT_MORE
 - BLE GAP APIs, 19
- GAP_ADTYPE_3D_INFO_DATA
 - BLE GAP APIs, 19
- GAP_ADTYPE_ADV_INTERVAL
 - BLE GAP APIs, 19
- GAP_ADTYPE_APPEARANCE
 - BLE GAP APIs, 19
- GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED
 - BLE GAP APIs, 19
- GAP_ADTYPE_FLAGS_GENERAL
 - BLE GAP APIs, 19
- GAP_ADTYPE_FLAGS_LIMITED
 - BLE GAP APIs, 20
- GAP_ADTYPE_FLAGS
 - BLE GAP APIs, 19
- GAP_ADTYPE_LE_BD_ADDR
 - BLE GAP APIs, 20
- GAP_ADTYPE_LE_ROLE
 - BLE GAP APIs, 20
- GAP_ADTYPE_LOCAL_NAME_COMPLETE
 - BLE GAP APIs, 20
- GAP_ADTYPE_LOCAL_NAME_SHORT
 - BLE GAP APIs, 20
- GAP_ADTYPE_MANUFACTURER_SPECIFIC
 - BLE GAP APIs, 20
- GAP_ADTYPE_OOB_CLASS_OF_DEVICE
 - BLE GAP APIs, 20
- GAP_ADTYPE_OOB_SIMPLE_PAIRING_HASHC
 - BLE GAP APIs, 20
- GAP_ADTYPE_OOB_SIMPLE_PAIRING_RANDR
 - BLE GAP APIs, 21
- GAP_ADTYPE_POWER_LEVEL
 - BLE GAP APIs, 21
- GAP_ADTYPE_PUBLIC_TARGET_ADDR
 - BLE GAP APIs, 21
- GAP_ADTYPE_RANDOM_TARGET_ADDR
 - BLE GAP APIs, 21
- GAP_ADTYPE_SERVICE_DATA_128BIT
 - BLE GAP APIs, 21
- GAP_ADTYPE_SERVICE_DATA_32BIT
 - BLE GAP APIs, 21
- GAP_ADTYPE_SERVICE_DATA
 - BLE GAP APIs, 21
- GAP_ADTYPE_SERVICES_LIST_128BIT
 - BLE GAP APIs, 21
- GAP_ADTYPE_SERVICES_LIST_16BIT
 - BLE GAP APIs, 22
- GAP_ADTYPE_SIGNED_DATA
 - BLE GAP APIs, 22
- GAP_ADTYPE_SIMPLE_PAIRING_HASHC_256
 - BLE GAP APIs, 22
- GAP_ADTYPE_SIMPLE_PAIRING_RANDR_256
 - BLE GAP APIs, 22
- GAP_ADTYPE_SLAVE_CONN_INTERVAL_RANGE
 - BLE GAP APIs, 22
- GAP_ADTYPE_SM_OOB_FLAG
 - BLE GAP APIs, 22
- GAP_ADTYPE_SM_TK
 - BLE GAP APIs, 22
- GAP_PUBLIC_ADDR
 - BLE GAP APIs, 22
- GAP_RAND_ADDR_NRPA
 - BLE GAP APIs, 23
- GAP_RAND_ADDR_RPA
 - BLE GAP APIs, 23
- GAP_RAND_ADDR_STATIC
 - BLE GAP APIs, 23
- GAP_SCAN_TYPE_ACTIVE
 - BLE GAP APIs, 23
- GAP_SCAN_TYPE_PASSIVE
 - BLE GAP APIs, 23
- GAP_TX_PWR_CURR_VAL
 - BLE GAP APIs, 23
- GAP_TX_PWR_MAX_VAL
 - BLE GAP APIs, 23
- GAPBOND_IO_CAP_DISPLAY_ONLY
 - BLE GAP APIs, 23
- GAPBOND_IO_CAP_DISPLAY_YES_NO
 - BLE GAP APIs, 24
- GAPBOND_IO_CAP_KEYBOARD_DISPLAY
 - BLE GAP APIs, 24
- GAPBOND_IO_CAP_KEYBOARD_ONLY
 - BLE GAP APIs, 24
- GAPBOND_IO_CAP_NO_INPUT_NO_OUTPUT
 - BLE GAP APIs, 24
- GAPBOND_PAIRING_MODE_INITIATE

- BLE GAP APIs, [24](#)
- GAPBOND_PAIRING_MODE_NO_PAIRING
 - BLE GAP APIs, [24](#)
- GAPBOND_PAIRING_MODE_WAIT_FOR_REQ
 - BLE GAP APIs, [24](#)
- GATT_CHAR_AGG_FORMAT_UUID
 - BLE GATT APIs, [43](#)
- GATT_CHAR_EXT_PROPS_UUID
 - BLE GATT APIs, [43](#)
- GATT_CHAR_FORMAT_UUID
 - BLE GATT APIs, [43](#)
- GATT_CHAR_USER_DESC_UUID
 - BLE GATT APIs, [44](#)
- GATT_CHARACTERISTIC_UUID
 - BLE GATT APIs, [44](#)
- GATT_CLIENT_CHAR_CFG_UUID
 - BLE GATT APIs, [44](#)
- GATT_EXT_REPORT_REF_UUID
 - BLE GATT APIs, [44](#)
- GATT_INCLUDE_UUID
 - BLE GATT APIs, [44](#)
- GATT_PRIMARY_SERVICE_UUID
 - BLE GATT APIs, [44](#)
- GATT_REPORT_REF_UUID
 - BLE GATT APIs, [44](#)
- GATT_SECONDARY_SERVICE_UUID
 - BLE GATT APIs, [44](#)
- GATT_SERV_CHAR_CFG_UUID
 - BLE GATT APIs, [45](#)
- GATT_VALID_RANGE_UUID
 - BLE GATT APIs, [45](#)
- gcCharAggregateUuid
 - BLE GATT APIs, [68](#)
- gcCharExtPropUuid
 - BLE GATT APIs, [68](#)
- gcCharFormatUuid
 - BLE GATT APIs, [69](#)
- gcCharUserDescUuid
 - BLE GATT APIs, [69](#)
- gcCharacteristicUuid
 - BLE GATT APIs, [68](#)
- gcClientCharConfigUuid
 - BLE GATT APIs, [69](#)
- gcExtReportRefUuid
 - BLE GATT APIs, [69](#)
- gcIncludeUuid
 - BLE GATT APIs, [69](#)
- gcPrimaryServiceUuid
 - BLE GATT APIs, [69](#)
- gcReportRefUuid
 - BLE GATT APIs, [69](#)
- gcSecondaryServiceUuid
 - BLE GATT APIs, [69](#)
- gcServerCharConfigUuid
 - BLE GATT APIs, [70](#)
- gcValidRangeUuid
 - BLE GATT APIs, [70](#)
- got_ip
 - wifi_event_info_t, [214](#)
- group_cipher
 - wifi_scan_info_t, [222](#)
- handle
 - LE_CM_MSG_SET_DISCONNECT_CFM_T, [154](#)
 - LE_GATT_ATTR_T, [160](#)
 - LE_GATT_MSG_ACCESS_READ_IND_T, [162](#)
 - LE_GATT_MSG_ACCESS_WRITE_IND_T, [163](#)
 - LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_↵
IND_T, [164](#)
 - LE_GATT_MSG_CHARACTERISTIC_DECL_IN↵
FO_IND_T, [165](#)
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IND↵
_T, [167](#)
 - LE_GATT_MSG_CONFIRMATION_CFM_T, [168](#)
 - LE_GATT_MSG_FIND_ALL_CHAR_DESC_CF↵
M_T, [171](#)
 - LE_GATT_MSG_FIND_ALL_PRIMARY_SERVI↵
CE_CFM_T, [172](#)
 - LE_GATT_MSG_FIND_CHARACTERISTIC_CF↵
M_T, [173](#)
 - LE_GATT_MSG_FIND_INCLUDED_SERVICE_↵
CFM_T, [174](#)
 - LE_GATT_MSG_FIND_PRIMARY_SERVICE_B↵
Y_UUID_CFM_T, [175](#)
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_I↵
ND_T, [176](#)
 - LE_GATT_MSG_INDICATE_IND_T, [177](#)
 - LE_GATT_MSG_NOTIFY_CFM_T, [178](#)
 - LE_GATT_MSG_NOTIFY_IND_T, [179](#)
 - LE_GATT_MSG_PREPARE_WRITE_RELIABL↵
E_CFM_T, [181](#)
 - LE_GATT_MSG_READ_CHAR_VAL_BY_UUID↵
_CFM_T, [182](#)
 - LE_GATT_MSG_READ_CHARACTERISTIC_V↵
ALUE_CFM_T, [183](#)
 - LE_GATT_MSG_READ_LONG_CHAR_VAL_C↵
FM_T, [184](#)
 - LE_GATT_MSG_SIGNED_WRITE_CFM_T, [187](#)
 - LE_GATT_MSG_WRITE_CHAR_VAL_RELIAB↵
LE_CFM_T, [188](#)
 - LE_GATT_MSG_WRITE_CHAR_VALUE_CFM↵
_T, [189](#)
 - LE_GATT_MSG_WRITE_LONG_CHAR_VALU↵
E_CFM_T, [190](#)
 - LE_GATT_MSG_WRITE_NO_RSP_CFM_T, [191](#)
- hap_ap_info
 - hap_control_t, [136](#)
- hap_bitvector
 - hap_control_t, [136](#)
- hap_control_t, [136](#)
 - hap_ap_info, [136](#)
 - hap_bitvector, [136](#)
 - hap_en, [136](#)
 - hap_final_index, [137](#)
 - hap_index, [137](#)
 - hap_ssid, [137](#)
- hap_en

- hap_control_t, 136
- hap_final_index
 - hap_control_t, 137
- hap_index
 - hap_control_t, 137
- hap_ssid
 - hap_control_t, 137
- hid_ssid
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 201
 - wifi_auto_connect_info_t, 211
- iArgc
 - T_RfCmd, 204
- INCLUDE_DECL_UUID128
 - BLE GATT APIs, 45
- INCLUDE_DECL_UUID128_ATTR_VAL
 - BLE GATT APIs, 45
- INCLUDE_DECL_UUID16_ATTR_VAL
 - BLE GATT APIs, 45
- INCLUDE_DECL_UUINT16
 - BLE GATT APIs, 45
- identifier
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 154
- interval
 - LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T, 142
 - LE_GAP_SCAN_PARAM_T, 159
- interval_max
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 155
 - LE_GAP_ADVERTISING_PARAM_T, 157
 - LE_GAP_CONN_PARAM_T, 158
- interval_min
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 155
 - LE_GAP_ADVERTISING_PARAM_T, 157
 - LE_GAP_CONN_PARAM_T, 158
- ip_changed
 - wifi_event_sta_got_ip_t, 217
- itv_max
 - LE_CM_MSG_CONN_PARA_REQ_T, 141
 - LE_CONN_PARA_T, 156
- itv_min
 - LE_CM_MSG_CONN_PARA_REQ_T, 141
 - LE_CONN_PARA_T, 156
- keypress
 - LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T, 198
- LE_ATT_MSG_BASE
 - BLE MSG APIs, 72
- LE_ATT_UUID_SIZE
 - BLE GATT APIs, 45
- LE_BT_ADDR_T, 137
 - addr, 137
 - type, 137
- LE_CM_CONNECTION_COMPLETE_IND_T, 138
 - conn_hdl, 138
 - conn_interval, 138
 - conn_latency, 138
 - dev_id, 138
 - peer_addr, 139
 - peer_addr_type, 139
 - role, 139
 - status, 139
 - supervision_timeout, 139
- LE_CM_MSG_ADD_TO_RESOLVING_LIST_CFM_T
 - BLE CM APIs, 11
- LE_CM_MSG_ADD_TO_WHITE_LIST_CFM_T
 - BLE CM APIs, 11
- LE_CM_MSG_ADVERTISE_REPORT_IND_T, 139
 - addr, 140
 - addr_type, 140
 - data, 140
 - event_type, 140
 - len, 140
 - rsssi, 140
- LE_CM_MSG_BASE
 - BLE MSG APIs, 72
- LE_CM_MSG_CANCEL_CONNECTION_CFM_T
 - BLE CM APIs, 11
- LE_CM_MSG_CLEAR_RESOLVING_LIST_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_CLEAR_WHITE_LIST_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_CONN_PARA_REQ_T, 140
 - conn_hdl, 141
 - itv_max, 141
 - itv_min, 141
 - latency, 141
 - sv_tmo, 141
- LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T, 141
 - conn_hdl, 142
 - interval, 142
 - latency, 142
 - status, 142
 - supervision_timeout, 142
- LE_CM_MSG_CREATE_CONNECTION_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 142
 - conn_hdl, 143
 - max_rx_octets, 143
 - max_rx_time, 143
 - max_tx_octets, 143
 - max_tx_time, 143
- LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 143
 - direct_addr, 144
 - direct_addr_type, 144
 - peer_addr, 144
 - peer_addr_type, 144
 - rsssi, 144
- LE_CM_MSG_DISCONNECT_COMPLETE_IND_T, 144
 - conn_hdl, 145
 - reason, 145
 - status, 145

- LE_CM_MSG_ENCRYPTION_CHANGE_IND_T, 145
 - conn_hdl, 145
 - devid, 146
 - enabled, 146
 - status, 146
- LE_CM_MSG_ENCRYPTION_REFRESH_IND_T, 146
 - conn_hdl, 146
 - devid, 146
 - enabled, 147
 - status, 147
- LE_CM_MSG_ENTER_ADVERTISING_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_ENTER_SCANNING_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_EXIT_ADVERTISING_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_EXIT_SCANNING_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_INIT_COMPLETE_CFM_T, 147
 - status, 147
- LE_CM_MSG_LTK_REQ_IND_T, 147
 - conn_hdl, 148
 - devid, 148
 - ediv, 148
 - rand, 148
- LE_CM_MSG_READ_ADV_TX_POWER_CFM_T, 148
 - pwr_level, 149
 - status, 149
- LE_CM_MSG_READ_BD_ADDR_CFM_T, 149
 - bd_addr, 149
 - status, 149
- LE_CM_MSG_READ_CHANNEL_MAP_CFM_T, 150
 - ch_map, 150
 - conn_hdl, 150
 - status, 150
- LE_CM_MSG_READ_RESOLVING_LIST_SIZE_CFM_T, 150
 - size, 150
 - status, 151
- LE_CM_MSG_READ_RSSI_CFM_T, 151
 - conn_hdl, 151
 - rssi, 151
 - status, 151
- LE_CM_MSG_READ_TX_POWER_CFM_T, 152
 - conn_hdl, 152
 - status, 152
 - tx_power, 152
- LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM_T, 152
 - size, 152
 - status, 153
- LE_CM_MSG_REMOVE_FROM_RESOLVING_LIST_CFM_T
 - BLE CM APIs, 12
- LE_CM_MSG_REMOVE_FROM_WHITE_LIST_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_ADVERTISING_DATA_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_ADVERTISING_PARAMS_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_CHANNEL_MAP_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_DATA_LENGTH_CFM_T, 153
 - conn_hdl, 153
 - status, 153
- LE_CM_MSG_SET_DISCONNECT_CFM_T, 153
 - handle, 154
 - status, 154
- LE_CM_MSG_SET_RANDOM_ADDRESS_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_RPA_TIMEOUT_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_SCAN_PARAMS_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SET_SCAN_RSP_DATA_CFM_T
 - BLE CM APIs, 13
- LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 154
 - conn_hdl, 154
 - identifier, 154
 - interval_max, 155
 - interval_min, 155
 - slave_latency, 155
 - timeout_multiplier, 155
- LE_CM_REQ_STATUS_T, 155
 - status, 155
- LE_CONN_PARA_T, 156
 - itv_max, 156
 - itv_min, 156
 - latency, 156
 - sv_timeout, 156
- LE_GAP_ADV_MAX_SIZE
 - BLE GAP APIs, 24
- LE_GAP_ADVERTISING_PARAM_T, 157
 - channel_map, 157
 - filter_policy, 157
 - interval_max, 157
 - interval_min, 157
 - own_addr_type, 157
 - peer_addr, 158
 - peer_addr_type, 158
 - type, 158
- LE_GAP_CONN_PARAM_T, 158
 - interval_max, 158
 - interval_min, 158
 - latency, 159
 - supervision_timeout, 159
- LE_GAP_SCAN_PARAM_T, 159
 - filter_policy, 159
 - interval, 159
 - own_addr_type, 159
 - type, 160
 - window, 160
- LE_GATT_ATTR_T, 160
 - format, 160

- handle, 160
- len, 161
- maxLen, 161
- pUuid, 161
- pVal, 161
- permit, 161
- LE_GATT_CHAR_PROP_AUTH
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_BCAST
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_EXT_PROP
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_IND
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_NTF
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_RD
 - BLE GATT APIs, 46
- LE_GATT_CHAR_PROP_WR_NO_RESP
 - BLE GATT APIs, 47
- LE_GATT_CHAR_PROP_WR
 - BLE GATT APIs, 46
- LE_GATT_CLIENT_CFG_INDICATION
 - BLE GATT APIs, 47
- LE_GATT_CLIENT_CFG_NOTIFICATION
 - BLE GATT APIs, 47
- LE_GATT_EXT_PROP_RELIABLE_WR
 - BLE GATT APIs, 47
- LE_GATT_EXT_PROP_WR_AUX
 - BLE GATT APIs, 47
- LE_GATT_FLAG_PREPARE_WRITE
 - BLE GATT APIs, 47
- LE_GATT_FLAG_WRITE_CMD
 - BLE GATT APIs, 47
- LE_GATT_FLAG_WRITE_REQ
 - BLE GATT APIs, 47
- LE_GATT_MSG_ACCESS_READ_IND_T, 161
 - conn_hdl, 162
 - devid, 162
 - handle, 162
 - offset, 162
- LE_GATT_MSG_ACCESS_WRITE_IND_T, 162
 - conn_hdl, 162
 - devid, 163
 - flag, 163
 - handle, 163
 - len, 163
 - offset, 163
 - pVal, 163
- LE_GATT_MSG_BASE
 - BLE MSG APIs, 72
- LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T, 163
 - conn_hdl, 164
 - devid, 164
 - format, 164
 - handle, 164
 - uuid, 164
- LE_GATT_MSG_CHARACTERISTIC_DECL_INFO_IND_T, 164
 - conn_hdl, 165
 - devid, 165
 - format, 165
 - handle, 165
 - property, 165
 - uuid, 165
 - val_hdl, 166
- LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T, 166
 - att_err, 166
 - conn_hdl, 166
 - devid, 166
 - handle, 167
 - len, 167
 - offset, 167
 - val, 167
- LE_GATT_MSG_CONFIRMATION_CFM_T, 167
 - conn_hdl, 167
 - devid, 168
 - handle, 168
- LE_GATT_MSG_EXCHANGE_MTU_CFM_T, 168
 - conn_hdl, 168
 - current_rx_mtu, 168
 - devid, 168
- LE_GATT_MSG_EXCHANGE_MTU_IND_T, 169
 - client_rx_mtu, 169
 - conn_hdl, 169
 - devid, 169
- LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T, 169
 - att_err, 170
 - conn_hdl, 170
 - devid, 170
 - err_hdl, 170
 - status, 170
- LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T, 170
 - att_err, 171
 - conn_hdl, 171
 - devid, 171
 - handle, 171
 - status, 171
- LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T, 171
 - att_err, 172
 - conn_hdl, 172
 - devid, 172
 - handle, 172
 - status, 172
- LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T, 172
 - att_err, 173
 - conn_hdl, 173
 - devid, 173
 - handle, 173
 - status, 173

- LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T, 173
 - att_err, 174
 - conn_hdl, 174
 - devid, 174
 - handle, 174
 - status, 174
- LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T, 174
 - att_err, 175
 - conn_hdl, 175
 - devid, 175
 - handle, 175
 - status, 175
- LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T, 175
 - conn_hdl, 176
 - devid, 176
 - end_hdl, 176
 - format, 176
 - handle, 176
 - start_hdl, 176
 - uuid, 177
- LE_GATT_MSG_INDICATE_IND_T, 177
 - conn_hdl, 177
 - devid, 177
 - handle, 177
 - len, 177
 - val, 178
- LE_GATT_MSG_NOTIFY_CFM_T, 178
 - conn_hdl, 178
 - devid, 178
 - handle, 178
 - status, 178
- LE_GATT_MSG_NOTIFY_IND_T, 179
 - conn_hdl, 179
 - devid, 179
 - handle, 179
 - len, 179
 - val, 179
- LE_GATT_MSG_OPERATION_TIMEOUT_T, 180
 - att_op, 180
 - conn_hdl, 180
 - devid, 180
- LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T, 180
 - att_err, 181
 - conn_hdl, 181
 - devid, 181
 - handle, 181
 - status, 181
- LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T, 181
 - att_err, 182
 - conn_hdl, 182
 - devid, 182
 - handle, 182
 - status, 182
- LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T, 182
 - att_err, 183
 - conn_hdl, 183
 - devid, 183
 - handle, 183
 - status, 183
- LE_GATT_MSG_READ_LONG_CHAR_VAL_CFM_T, 183
 - att_err, 184
 - conn_hdl, 184
 - devid, 184
 - handle, 184
 - status, 184
- LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFM_T, 184
 - att_err, 185
 - conn_hdl, 185
 - devid, 185
 - err_hdl, 185
 - len, 185
 - status, 185
 - val, 186
- LE_GATT_MSG_SERVICE_INFO_IND_T, 186
 - conn_hdl, 186
 - devid, 186
 - end_hdl, 186
 - format, 186
 - start_hdl, 187
 - uuid, 187
- LE_GATT_MSG_SIGNED_WRITE_CFM_T, 187
 - conn_hdl, 187
 - devid, 187
 - handle, 187
 - status, 188
- LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T, 188
 - att_err, 188
 - conn_hdl, 188
 - devid, 188
 - handle, 188
 - status, 189
- LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T, 189
 - att_err, 189
 - conn_hdl, 189
 - devid, 189
 - handle, 189
 - status, 190
- LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T, 190
 - att_err, 190
 - conn_hdl, 190
 - devid, 190
 - handle, 190
 - status, 191
- LE_GATT_MSG_WRITE_NO_RSP_CFM_T, 191
 - conn_hdl, 191
 - devid, 191

- handle, [191](#)
- status, [191](#)
- LE_GATT_PERM_AUTH_READABLE
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_AUTH_WRITABLE
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_NONE
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_READ
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_RELIABLE_WRITE
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_WRITE_CMD
 - BLE GATT APIs, [48](#)
- LE_GATT_PERM_WRITE_REQ
 - BLE GATT APIs, [48](#)
- LE_GATT_PERMIT_AUTHEN_READ
 - BLE GATT APIs, [48](#)
- LE_GATT_PERMIT_AUTHEN_WRITE
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_AUTHOR_READ
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_AUTHOR_WRITE
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_ENCRYPT_READ
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_ENCRYPT_WRITE
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_READABLE
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_READ
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_SC_AUTHEN_READ
 - BLE GATT APIs, [49](#)
- LE_GATT_PERMIT_SC_AUTHEN_WRITE
 - BLE GATT APIs, [50](#)
- LE_GATT_PERMIT_WRITABLE
 - BLE GATT APIs, [50](#)
- LE_GATT_PERMIT_WRITE
 - BLE GATT APIs, [50](#)
- LE_GATT_SERVICE_T, [192](#)
 - endHdl, [192](#)
 - pAttr, [192](#)
 - startHdl, [192](#)
 - svc_id, [192](#)
- LE_HCI_MSG_BASE
 - BLE MSG APIs, [73](#)
- LE_L2CAP_MSG_BASE
 - BLE MSG APIs, [73](#)
- LE_MAX_BOND_COUNT
 - BLE SMP APIs, [84](#)
- LE_SM_IO_CAP_DISP_ONLY
 - BLE SMP APIs, [84](#)
- LE_SM_IO_CAP_DISP_YES_NO
 - BLE SMP APIs, [84](#)
- LE_SM_IO_CAP_KEYBOARD_DISP
 - BLE SMP APIs, [84](#)
- LE_SM_IO_CAP_KEYBOARD_ONLY
 - BLE SMP APIs, [85](#)
- LE_SM_IO_CAP_NO_IO
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_MITM_NO
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_MITM_YES
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_OOB_NO
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_OOB_YES
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_SC_NO
 - BLE SMP APIs, [85](#)
- LE_SM_PAIR_SC_YES
 - BLE SMP APIs, [85](#)
- LE_SMP_MSG_BASE
 - BLE MSG APIs, [73](#)
- LE_SMP_MSG_ENCRYPTION_CHANGE_IND_T, [193](#)
 - conn_hdl, [193](#)
 - enable, [193](#)
- LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T, [193](#)
 - conn_hdl, [193](#)
 - status, [193](#)
- LE_SMP_MSG_OOB_DATA_REQUEST_IND_T, [194](#)
 - conn_hdl, [194](#)
- LE_SMP_MSG_PAIRING_ACTION_IND_T, [194](#)
 - action, [194](#)
 - conn_hdl, [194](#)
 - lost_bond, [195](#)
 - sc, [195](#)
- LE_SMP_MSG_PAIRING_COMPLETE_IND_T, [195](#)
 - authenticated, [195](#)
 - bonded, [195](#)
 - conn_hdl, [195](#)
 - peer_id_addr, [196](#)
 - sc, [196](#)
 - status, [196](#)
- LE_SMP_MSG_PASSKEY_DISPLAY_IND_T, [196](#)
 - conn_hdl, [196](#)
 - passkey, [196](#)
- LE_SMP_MSG_PASSKEY_INPUT_IND_T, [197](#)
 - conn_hdl, [197](#)
- LE_SMP_MSG_SC_OOB_DATA_REQUEST_IND_T, [197](#)
 - conn_hdl, [197](#)
- LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T, [198](#)
 - bondable, [198](#)
 - conn_hdl, [198](#)
 - keypress, [198](#)
 - mitm, [198](#)
 - sc, [198](#)
- LE_SMP_MSG_USER_CONFIRM_IND_T, [199](#)
 - confirm_num, [199](#)
 - conn_hdl, [199](#)
- LE_SMP_SC_OOB_DATA_T, [199](#)
 - confirm, [199](#)

- rand, [199](#)
- LE_SYS_MSG_BASE
 - BLE MSG APIs, [73](#)
- LE_SYS_MSG_BUF_OVERFLOW_T, [200](#)
 - conn_hdl, [200](#)
- latency
 - LE_CM_MSG_CONN_PARA_REQ_T, [141](#)
 - LE_CM_MSG_CONN_UPDATE_COMPLETE_I↔ND_T, [142](#)
 - LE_CONN_PARA_T, [156](#)
 - LE_GAP_CONN_PARAM_T, [159](#)
- latest_beacon_rx_time
 - auto_conn_info_t, [132](#)
 - mw_wifi_auto_connect_ap_info_t, [202](#)
 - wifi_auto_connect_info_t, [211](#)
- LeCancelAllMessage
 - BLE MSG APIs, [76](#)
- LeCancelAllSubMessage
 - BLE MSG APIs, [77](#)
- LeCancelFirstMessage
 - BLE MSG APIs, [77](#)
- LeCancelFirstSubMessage
 - BLE MSG APIs, [77](#)
- LeCmInit
 - BLE CM APIs, [15](#)
- LeGapAddToResolvingList
 - BLE GAP APIs, [25](#)
- LeGapAddToWhiteList
 - BLE GAP APIs, [25](#)
- LeGapAdvertisingEnable
 - BLE GAP APIs, [25](#)
- LeGapCentralConnectReq
 - BLE GAP APIs, [26](#)
- LeGapCentralSetDataChannel
 - BLE GAP APIs, [26](#)
- LeGapClearResolvingList
 - BLE GAP APIs, [27](#)
- LeGapClearWhiteList
 - BLE GAP APIs, [27](#)
- LeGapConnParaRequestRsp
 - BLE GAP APIs, [27](#)
- LeGapConnUpdateRequest
 - BLE GAP APIs, [28](#)
- LeGapConnUpdateResponse
 - BLE GAP APIs, [28](#)
- LeGapConnectCancelReq
 - BLE GAP APIs, [27](#)
- LeGapDisconnectReq
 - BLE GAP APIs, [29](#)
- LeGapGenRandAddr
 - BLE GAP APIs, [29](#)
- LeGapGetBtAddr
 - BLE GAP APIs, [29](#)
- LeGapReadAdvChannelTxPower
 - BLE GAP APIs, [29](#)
- LeGapReadChannelMap
 - BLE GAP APIs, [30](#)
- LeGapReadResolvingListSize
 - BLE GAP APIs, [30](#)
- LeGapReadRssi
 - BLE GAP APIs, [30](#)
- LeGapReadTxPower
 - BLE GAP APIs, [31](#)
- LeGapReadWhiteListSize
 - BLE GAP APIs, [31](#)
- LeGapRemoveFromWhiteList
 - BLE GAP APIs, [31](#)
- LeGapScanningReq
 - BLE GAP APIs, [32](#)
- LeGapSetAdvData
 - BLE GAP APIs, [32](#)
- LeGapSetAdvParameter
 - BLE GAP APIs, [33](#)
- LeGapSetConnParameter
 - BLE GAP APIs, [33](#)
- LeGapSetDataChannelPduLen
 - BLE GAP APIs, [33](#)
- LeGapSetRandAddr
 - BLE GAP APIs, [34](#)
- LeGapSetRpaTimeout
 - BLE GAP APIs, [34](#)
- LeGapSetStaticAddr
 - BLE GAP APIs, [35](#)
- LeGattAccessReadRsp
 - BLE GATT APIs, [52](#)
- LeGattAccessWriteRsp
 - BLE GATT APIs, [52](#)
- LeGattChangeAttrVal
 - BLE GATT APIs, [53](#)
- LeGattCharValConfirmation
 - BLE GATT APIs, [53](#)
- LeGattCharValIndicate
 - BLE GATT APIs, [54](#)
- LeGattCharValNotify
 - BLE GATT APIs, [54](#)
- LeGattExchangeMtuReq
 - BLE GATT APIs, [55](#)
- LeGattExchangeMtuRsp
 - BLE GATT APIs, [55](#)
- LeGattExecuteWriteCharValReliable
 - BLE GATT APIs, [55](#)
- LeGattFindAllCharDescriptor
 - BLE GATT APIs, [56](#)
- LeGattFindAllCharacteristic
 - BLE GATT APIs, [56](#)
- LeGattFindAllPrimaryService
 - BLE GATT APIs, [57](#)
- LeGattFindCharacteristicByUuid
 - BLE GATT APIs, [57](#)
- LeGattFindIncludedService
 - BLE GATT APIs, [58](#)
- LeGattFindPrimaryServiceByUuid
 - BLE GATT APIs, [58](#)
- LeGattGetAttrHandle
 - BLE GATT APIs, [58](#)
- LeGattGetAttrVal

- BLE GATT APIs, [59](#)
- LeGattGetAttrValLen
 - BLE GATT APIs, [59](#)
- LeGattGetAttrValMaxLen
 - BLE GATT APIs, [61](#)
- LeGattInit
 - BLE GATT APIs, [61](#)
- LeGattModifyAttrVal
 - BLE GATT APIs, [62](#)
- LeGattPrepareWriteCharValReliable
 - BLE GATT APIs, [62](#)
- LeGattReadCharValByUuid
 - BLE GATT APIs, [63](#)
- LeGattReadCharValue
 - BLE GATT APIs, [63](#)
- LeGattReadLongCharVal
 - BLE GATT APIs, [64](#)
- LeGattReadMultipleCharVal
 - BLE GATT APIs, [64](#)
- LeGattRegisterIncludeService
 - BLE GATT APIs, [64](#)
- LeGattRegisterService
 - BLE GATT APIs, [65](#)
- LeGattSignedWriteNoRsp
 - BLE GATT APIs, [65](#)
- LeGattStopCurrentProcedure
 - BLE GATT APIs, [66](#)
- LeGattWriteCharVal
 - BLE GATT APIs, [66](#)
- LeGattWriteCharValReliable
 - BLE GATT APIs, [67](#)
- LeGattWriteLongCharVal
 - BLE GATT APIs, [67](#)
- LeGattWriteNoRsp
 - BLE GATT APIs, [68](#)
- LeGetSubMsgId
 - BLE MSG APIs, [78](#)
- LeHostCreateTask
 - BLE MSG APIs, [78](#)
- LeHostMessageLoop
 - BLE MSG APIs, [79](#)
- LeSendMessage
 - BLE MSG APIs, [79](#)
- LeSendMessageAfter
 - BLE MSG APIs, [79](#)
- LeSendMessageUnlock
 - BLE MSG APIs, [80](#)
- LeSendSubMessage
 - BLE MSG APIs, [80](#)
- LeSendSubMessageAfter
 - BLE MSG APIs, [81](#)
- LeSendSubMessageUnlock
 - BLE MSG APIs, [81](#)
- LeSetScanParameter
 - BLE GAP APIs, [35](#)
- LeSetScanRspData
 - BLE GAP APIs, [35](#)
- LeSmpInit
 - BLE SMP APIs, [87](#)
- LeSmpOobAuthDataRsp
 - BLE SMP APIs, [87](#)
- LeSmpOobPresent
 - BLE SMP APIs, [87](#)
- LeSmpPasskeyInput
 - BLE SMP APIs, [88](#)
- LeSmpScOobComputeConfirmVal
 - BLE SMP APIs, [88](#)
- LeSmpScOobDataRsp
 - BLE SMP APIs, [88](#)
- LeSmpSecurityReq
 - BLE SMP APIs, [89](#)
- LeSmpSecurityRsp
 - BLE SMP APIs, [89](#)
- LeSmpSetDefaultConfig
 - BLE SMP APIs, [90](#)
- LeSmpUserConfirmRsp
 - BLE SMP APIs, [90](#)
- len
 - LE_CM_MSG_ADVERTISE_REPORT_IND_↔
T, [140](#)
 - LE_GATT_ATTR_T, [161](#)
 - LE_GATT_MSG_ACCESS_WRITE_IND_T, [163](#)
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IND_↔
T, [167](#)
 - LE_GATT_MSG_INDICATE_IND_T, [177](#)
 - LE_GATT_MSG_NOTIFY_IND_T, [179](#)
 - LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_↔
L_CFM_T, [185](#)
- length
 - event_msg_t, [136](#)
- lost_bond
 - LE_SMP_MSG_PAIRING_ACTION_IND_T, [195](#)
- MESSAGE_ALLOCATE
 - BLE MSG APIs, [73](#)
- MESSAGE_BULID
 - BLE MSG APIs, [73](#)
- MESSAGE_DATA_BULID
 - BLE MSG APIs, [73](#)
- MESSAGE_OFFSET
 - BLE MSG APIs, [74](#)
- MESSAGEID
 - BLE MSG APIs, [74](#)
- MESSAGE
 - BLE MSG APIs, [74](#)
- MSGLOCK
 - BLE MSG APIs, [75](#)
- MSGSUBID
 - BLE MSG APIs, [75](#)
- MSGTIMER
 - BLE MSG APIs, [75](#)
- magic
 - wifi_init_config_t, [219](#)
- max
 - wifi_active_scan_time_t, [207](#)
- max_connection
 - wifi_ap_config_t, [209](#)

- max_rx_octets
 - LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 143
- max_rx_time
 - LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 143
- max_save_num
 - auto_connect_cfg_t, 134
 - MwFimAutoConnectCFG_t, 203
- max_tx_octets
 - LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 143
- max_tx_time
 - LE_CM_MSG_DATA_LEN_CHANGE_IND_T, 143
- maxLen
 - LE_GATT_ATTR_T, 161
- min
 - wifi_active_scan_time_t, 207
- mitm
 - LE_SMP_MSG_SLAVE_SECURITY_REQUEST_IND_T, 198
- MsgData
 - BLE MSG APIs, 75
- MsgLock
 - BLE MSG APIs, 75
- mw_wifi_auto_connect_ap_info_t, 200
 - ap_channel, 201
 - beacon_interval, 201
 - bssid, 201
 - capabilities, 201
 - dtim_prod, 201
 - fast_connect, 201
 - free_ocpy, 201
 - hid_ssid, 201
 - latest_beacon_rx_time, 202
 - passphrase, 202
 - psk, 202
 - rsn_ie, 202
 - rsi, 202
 - ssid, 202
 - supported_rates, 202
 - wpa_data, 202
 - wpa_ie, 203
- MwFimAutoConnectCFG_t, 203
 - flag, 203
 - front, 203
 - max_save_num, 203
 - rear, 203
 - targetIdx, 204
- num
 - wifi_scan_list_t, 223
- number
 - wifi_event_sta_scan_done_t, 217
- offset
 - LE_GATT_MSG_ACCESS_READ_IND_T, 162
 - LE_GATT_MSG_ACCESS_WRITE_IND_T, 163
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T, 167
- own_addr_type
 - LE_GAP_ADVERTISING_PARAM_T, 157
- LE_GAP_SCAN_PARAM_T, 159
- pAttr
 - LE_GATT_SERVICE_T, 192
- pFCInfo
 - auto_connect_cfg_t, 134
- pParam
 - T_RfEvt, 205
- PRIMARY_SERVICE_DECL_UUID128
 - BLE GATT APIs, 50
- PRIMARY_SERVICE_DECL_UUID16
 - BLE GATT APIs, 50
- pUuid
 - LE_GATT_ATTR_T, 161
- pVal
 - LE_GATT_ATTR_T, 161
 - LE_GATT_MSG_ACCESS_WRITE_IND_T, 163
- pairwise_cipher
 - wifi_scan_info_t, 222
- param
 - event_msg_t, 136
- passive
 - wifi_scan_time_t, 223
- passkey
 - LE_SMP_MSG_PASSKEY_DISPLAY_IND_T, 196
- passphrase
 - auto_conn_info_t, 132
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_auto_connect_info_t, 211
- password
 - wifi_ap_config_t, 209
 - wifi_sta_config_t, 224
- password_length
 - wifi_ap_config_t, 209
 - wifi_sta_config_t, 225
- peer_addr
 - LE_CM_CONNECTION_COMPLETE_IND_T, 139
 - LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 144
 - LE_GAP_ADVERTISING_PARAM_T, 158
- peer_addr_type
 - LE_CM_CONNECTION_COMPLETE_IND_T, 139
 - LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 144
 - LE_GAP_ADVERTISING_PARAM_T, 158
- peer_id_addr
 - LE_SMP_MSG_PAIRING_COMPLETE_IND_T, 196
- permit
 - LE_GATT_ATTR_T, 161
- property
 - LE_GATT_MSG_CHARACTERISTIC_DECL_IND_T, 165
- psk
 - auto_conn_info_t, 133
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_auto_connect_info_t, 211
- pwr_level

- LE_CM_MSG_READ_ADV_TX_POWER_CFM↔
_T, 149
- rand
 - LE_CM_MSG_LTK_REQ_IND_T, 148
 - LE_SMP_SC_OOB_DATA_T, 199
- rear
 - auto_connect_cfg_t, 134
 - MwFimAutoConnectCFG_t, 203
- reason
 - LE_CM_MSG_DISCONNECT_COMPLETE_IN↔
D_T, 145
 - wifi_event_sta_disconnected_t, 216
- retryCount
 - auto_connect_cfg_t, 135
- role
 - LE_CM_CONNECTION_COMPLETE_IND_T, 139
- rsn_ie
 - auto_conn_info_t, 133
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_auto_connect_info_t, 212
- rssi
 - auto_conn_info_t, 133
 - LE_CM_MSG_ADVERTISE_REPORT_IND↔
T, 140
 - LE_CM_MSG_DIRECT_ADV_REPORT_IND_T, 144
 - LE_CM_MSG_READ_RSSI_CFM_T, 151
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_auto_connect_info_t, 212
 - wifi_fast_scan_threshold_t, 218
 - wifi_scan_info_t, 222
- SECONDARY_SERVICE_DECL_UUID128
 - BLE GATT APIs, 50
- SECONDARY_SERVICE_DECL_UUID16
 - BLE GATT APIs, 50
- saArgv
 - T_RfCmd, 204
- sc
 - LE_SMP_MSG_PAIRING_ACTION_IND_T, 195
 - LE_SMP_MSG_PAIRING_COMPLETE_IND_T, 196
 - LE_SMP_MSG_SLAVE_SECURITY_REQUEST↔
T_IND_T, 198
- scan_done
 - wifi_event_info_t, 214
- scan_id
 - wifi_event_sta_scan_done_t, 217
- scan_method
 - wifi_sta_config_t, 225
- scan_time
 - wifi_scan_config_t, 220
- scan_type
 - wifi_scan_config_t, 220
- show_hidden
 - wifi_scan_config_t, 220
- size
 - LE_CM_MSG_READ_RESOLVING_LIST_SIZE↔
_CFM_T, 150
 - LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM↔
_T, 152
- slave_latency
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 155
- sort_method
 - wifi_sta_config_t, 225
- ssid
 - auto_conn_info_t, 133
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_ap_config_t, 209
 - wifi_auto_connect_info_t, 212
 - wifi_event_sta_connected_t, 215
 - wifi_event_sta_disconnected_t, 216
 - wifi_scan_config_t, 220
 - wifi_scan_info_t, 222
 - wifi_sta_config_t, 225
- ssid_hidden
 - wifi_ap_config_t, 209
- ssid_len
 - wifi_event_sta_connected_t, 215
 - wifi_event_sta_disconnected_t, 216
- ssid_length
 - wifi_ap_config_t, 209
 - wifi_scan_info_t, 222
 - wifi_sta_config_t, 225
- sta_config
 - wifi_config_t, 213
- start_hdl
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_I↔
ND_T, 176
 - LE_GATT_MSG_SERVICE_INFO_IND_T, 187
- startHdl
 - LE_GATT_SERVICE_T, 192
- status
 - LE_CM_CONNECTION_COMPLETE_IND_T, 139
 - LE_CM_MSG_CONN_UPDATE_COMPLETE_I↔
ND_T, 142
 - LE_CM_MSG_DISCONNECT_COMPLETE_IN↔
D_T, 145
 - LE_CM_MSG_ENCRYPTION_CHANGE_IND_T, 146
 - LE_CM_MSG_ENCRYPTION_REFRESH_IND_T, 147
 - LE_CM_MSG_INIT_COMPLETE_CFM_T, 147
 - LE_CM_MSG_READ_ADV_TX_POWER_CFM↔
_T, 149
 - LE_CM_MSG_READ_BD_ADDR_CFM_T, 149
 - LE_CM_MSG_READ_CHANNEL_MAP_CFM_T, 150
 - LE_CM_MSG_READ_RESOLVING_LIST_SIZE↔
_CFM_T, 151
 - LE_CM_MSG_READ_RSSI_CFM_T, 151
 - LE_CM_MSG_READ_TX_POWER_CFM_T, 152
 - LE_CM_MSG_READ_WHITE_LIST_SIZE_CFM↔
_T, 153
 - LE_CM_MSG_SET_DATA_LENGTH_CFM_T,

- 153
- LE_CM_MSG_SET_DISCONNECT_CFM_T, 154
- LE_CM_REQ_STATUS_T, 155
- LE_GATT_MSG_EXECUTE_WRITE_RELIABLE_CFM_T, 170
- LE_GATT_MSG_FIND_ALL_CHAR_DESC_CFM_T, 171
- LE_GATT_MSG_FIND_ALL_PRIMARY_SERVICE_CFM_T, 172
- LE_GATT_MSG_FIND_CHARACTERISTIC_CFM_T, 173
- LE_GATT_MSG_FIND_INCLUDED_SERVICE_CFM_T, 174
- LE_GATT_MSG_FIND_PRIMARY_SERVICE_BY_UUID_CFM_T, 175
- LE_GATT_MSG_NOTIFY_CFM_T, 178
- LE_GATT_MSG_PREPARE_WRITE_RELIABLE_CFM_T, 181
- LE_GATT_MSG_READ_CHAR_VAL_BY_UUID_CFM_T, 182
- LE_GATT_MSG_READ_CHARACTERISTIC_VALUE_CFM_T, 183
- LE_GATT_MSG_READ_LONG_CHAR_VALUE_CFM_T, 184
- LE_GATT_MSG_READ_MULTIPLE_CHAR_VALUES_CFM_T, 185
- LE_GATT_MSG_SIGNED_WRITE_CFM_T, 188
- LE_GATT_MSG_WRITE_CHAR_VAL_RELIABLE_CFM_T, 189
- LE_GATT_MSG_WRITE_CHAR_VALUE_CFM_T, 190
- LE_GATT_MSG_WRITE_LONG_CHAR_VALUE_CFM_T, 191
- LE_GATT_MSG_WRITE_NO_RSP_CFM_T, 191
- LE_SMP_MSG_ENCRYPTION_REFRESH_IND_T, 193
- LE_SMP_MSG_PAIRING_COMPLETE_IND_T, 196
- wifi_event_sta_scan_done_t, 217
- supervision_timeout
 - LE_CM_MSG_CONN_UPDATE_COMPLETE_IND_T, 142
 - LE_GAP_CONN_PARAM_T, 159
- supervision_timeout
 - LE_CM_CONNECTION_COMPLETE_IND_T, 139
- supported_rates
 - auto_conn_info_t, 133
 - mw_wifi_auto_connect_ap_info_t, 202
 - wifi_auto_connect_info_t, 212
- sv_timeout
 - LE_CONN_PARAM_T, 156
- sv_tmo
 - LE_CM_MSG_CONN_PARAM_REQ_T, 141
- svc_id
 - LE_GATT_SERVICE_T, 192
- T_HOUR
 - BLE MSG APIs, 74
- T_MIN
 - BLE MSG APIs, 74
- T_RfCmd, 204
 - iArgc, 204
 - saArgv, 204
 - u32Type, 204
- T_RfEvt, 204
 - pParam, 205
 - u16RfMode, 205
 - u16RxCnt, 205
 - u16RxCrcOkCnt, 205
 - u32Freq, 205
 - u32Mode, 206
 - u32RfChannel, 206
 - u32Type, 206
 - u8Freq, 206
 - u8IpcEnable, 206
 - u8Len, 206
 - u8Pkt, 206
 - u8Reserved, 206
 - u8Status, 207
 - u8Unicast, 207
- T_SEC
 - BLE MSG APIs, 74
- TASKHANDLER
 - BLE MSG APIs, 75
- TASKPACK
 - BLE MSG APIs, 76
- TASK
 - BLE MSG APIs, 75
- targetIdx
 - auto_connect_cfg_t, 135
 - MwFimAutoConnectCFG_t, 204
- Task
 - BLE MSG APIs, 75
- threshold
 - wifi_sta_config_t, 225
- timeout_multiplier
 - LE_CM_MSG_SIGNAL_UPDATE_REQ_T, 155
- tx_power
 - LE_CM_MSG_READ_TX_POWER_CFM_T, 152
- type
 - LE_BT_ADDR_T, 137
 - LE_GAP_ADVERTISING_PARAM_T, 158
 - LE_GAP_SCAN_PARAM_T, 160
- u16RfMode
 - T_RfEvt, 205
- u16RxCnt
 - T_RfEvt, 205
- u16RxCrcOkCnt
 - T_RfEvt, 205
- u32Freq
 - T_RfEvt, 205
- u32Mode
 - T_RfEvt, 206
- u32RfChannel
 - T_RfEvt, 206
- u32Type
 - T_RfCmd, 204

- T_RfEvt, [206](#)
- u8Freq
 - T_RfEvt, [206](#)
- u8IpcEnable
 - T_RfEvt, [206](#)
- u8Len
 - T_RfEvt, [206](#)
- u8Pkt
 - T_RfEvt, [206](#)
- u8Reserved
 - T_RfEvt, [206](#)
- u8Status
 - T_RfEvt, [207](#)
- u8Unicast
 - T_RfEvt, [207](#)
- uFCAPNum
 - auto_connect_cfg_t, [135](#)
- uuid
 - LE_GATT_MSG_CHAR_DESCRIPTOR_INFO_IND_T, [164](#)
 - LE_GATT_MSG_CHARACTERISTIC_DECL_IND_T, [165](#)
 - LE_GATT_MSG_INCLUDE_SERVICE_INFO_IND_T, [177](#)
 - LE_GATT_MSG_SERVICE_INFO_IND_T, [187](#)
- val
 - LE_GATT_MSG_CHARACTERISTIC_VAL_IND_T, [167](#)
 - LE_GATT_MSG_INDICATE_IND_T, [178](#)
 - LE_GATT_MSG_NOTIFY_IND_T, [179](#)
 - LE_GATT_MSG_READ_MULTIPLE_CHAR_VAL_CFGM_T, [186](#)
- val_hdl
 - LE_GATT_MSG_CHARACTERISTIC_DECL_IND_T, [166](#)
- WIFI APIs, [91](#)
 - WIFI_BEACON_INTERVAL_LENGTH, [92](#)
 - WIFI_CAPABILITY_INFO_LENGTH, [92](#)
 - WIFI_LENGTH_802_11, [92](#)
 - WIFI_LENGTH_PASSPHRASE, [92](#)
 - WIFI_MAC_ADDRESS_LENGTH, [93](#)
 - WIFI_MAX_LENGTH_OF_SSID, [93](#)
 - WIFI_MAX_SCAN_AP_NUM, [93](#)
 - WIFI_MAX_SUPPORTED_RATES, [93](#)
 - wifi_event_notify_cb_t, [93](#)
 - wifi_event_process_handler, [94](#)
 - wifi_install_default_event_handlers, [94](#)
 - wifi_register_event_handler, [94](#)
- WIFI Common APIs, [96](#)
 - wifi_event_cb_t, [96](#)
 - wifi_event_loop_init, [97](#)
 - wifi_event_loop_send, [98](#)
 - wifi_event_loop_set_cb, [98](#)
 - wifi_event_process_handler, [99](#)
- WIFI STA APIs, [100](#)
 - wifi_auto_connect_del_ap_info, [103](#)
 - wifi_auto_connect_get_ap_info, [103](#)
 - wifi_auto_connect_get_ap_num, [104](#)
 - wifi_auto_connect_get_mode, [104](#)
 - wifi_auto_connect_init, [104](#)
 - wifi_auto_connect_reset, [104](#)
 - wifi_auto_connect_set_ap_num, [105](#)
 - wifi_auto_connect_set_mode, [105](#)
 - wifi_auto_connect_start, [106](#)
 - wifi_config_get_bandwidth, [106](#)
 - wifi_config_get_bssid, [106](#)
 - wifi_config_get_channel, [107](#)
 - wifi_config_get_dtim_interval, [107](#)
 - wifi_config_get_listen_interval, [108](#)
 - wifi_config_get_mac_address, [108](#)
 - wifi_config_get_opmode, [109](#)
 - wifi_config_get_skip_dtim, [109](#)
 - wifi_config_get_ssid, [109](#)
 - wifi_config_set_bandwidth, [110](#)
 - wifi_config_set_bssid, [110](#)
 - wifi_config_set_channel, [111](#)
 - wifi_config_set_dtim_interval, [111](#)
 - wifi_config_set_listen_interval, [112](#)
 - wifi_config_set_mac_address, [112](#)
 - wifi_config_set_opmode, [113](#)
 - wifi_config_set_skip_dtim, [113](#)
 - wifi_config_set_ssid, [113](#)
 - wifi_connection_connect, [115](#)
 - wifi_connection_disconnect_ap, [115](#)
 - wifi_connection_disconnect_sta, [115](#)
 - wifi_connection_get_rssi, [116](#)
 - wifi_connection_register_event_handler, [116](#)
 - wifi_connection_scan_start, [117](#)
 - wifi_connection_unregister_event_handler, [117](#)
 - wifi_deinit, [118](#)
 - wifi_event_handler_t, [102](#)
 - wifi_fast_connect_get_mode, [118](#)
 - wifi_fast_connect_set_mode, [119](#)
 - wifi_fast_connect_start, [119](#)
 - wifi_get_config, [119](#)
 - wifi_init, [120](#)
 - wifi_init_complete_cb_t, [102](#)
 - wifi_result_t, [103](#)
 - wifi_scan_get_ap_list, [120](#)
 - wifi_scan_get_ap_num, [121](#)
 - wifi_scan_get_ap_records, [121](#)
 - wifi_scan_scan_stop, [122](#)
 - wifi_scan_start, [122](#)
 - wifi_set_config, [122](#)
 - wifi_sta_get_ap_info, [123](#)
 - wifi_start, [123](#)
 - wifi_stop, [124](#)
- WIFI_BEACON_INTERVAL_LENGTH
 - WIFI APIs, [92](#)
- WIFI_CAPABILITY_INFO_LENGTH
 - WIFI APIs, [92](#)
- WIFI_LENGTH_802_11
 - WIFI APIs, [92](#)
- WIFI_LENGTH_PASSPHRASE
 - WIFI APIs, [92](#)

- WIFI_MAC_ADDRESS_LENGTH
 - WIFI APIs, [93](#)
- WIFI_MAX_LENGTH_OF_SSID
 - WIFI APIs, [93](#)
- WIFI_MAX_SCAN_AP_NUM
 - WIFI APIs, [93](#)
- WIFI_MAX_SUPPORTED_RATES
 - WIFI APIs, [93](#)
- wifi_active_scan_time_t, [207](#)
 - max, [207](#)
 - min, [207](#)
- wifi_ap_config_t, [208](#)
 - auth_mode, [208](#)
 - beacon_interval, [208](#)
 - channel, [208](#)
 - encrypt_type, [209](#)
 - max_connection, [209](#)
 - password, [209](#)
 - password_length, [209](#)
 - ssid, [209](#)
 - ssid_hidden, [209](#)
 - ssid_length, [209](#)
- wifi_auth_mode_t
 - Enumeration, [125](#)
- wifi_auto_connect_del_ap_info
 - WIFI STA APIs, [103](#)
- wifi_auto_connect_get_ap_info
 - WIFI STA APIs, [103](#)
- wifi_auto_connect_get_ap_num
 - WIFI STA APIs, [104](#)
- wifi_auto_connect_get_mode
 - WIFI STA APIs, [104](#)
- wifi_auto_connect_info_t, [210](#)
 - ap_channel, [210](#)
 - beacon_interval, [210](#)
 - bssid, [210](#)
 - capabilities, [211](#)
 - dtim_prod, [211](#)
 - fast_connect, [211](#)
 - free_ocpy, [211](#)
 - hid_ssid, [211](#)
 - latest_beacon_rx_time, [211](#)
 - passphrase, [211](#)
 - psk, [211](#)
 - rsn_ie, [212](#)
 - rsi, [212](#)
 - ssid, [212](#)
 - supported_rates, [212](#)
 - wpa_data, [212](#)
 - wpa_ie, [212](#)
- wifi_auto_connect_init
 - WIFI STA APIs, [104](#)
- wifi_auto_connect_reset
 - WIFI STA APIs, [104](#)
- wifi_auto_connect_set_ap_num
 - WIFI STA APIs, [105](#)
- wifi_auto_connect_set_mode
 - WIFI STA APIs, [105](#)
- wifi_auto_connect_start
 - WIFI STA APIs, [106](#)
- wifi_bandwidth_t
 - Enumeration, [126](#)
- wifi_cipher_type_t
 - Enumeration, [126](#)
- wifi_config_get_bandwidth
 - WIFI STA APIs, [106](#)
- wifi_config_get_bssid
 - WIFI STA APIs, [106](#)
- wifi_config_get_channel
 - WIFI STA APIs, [107](#)
- wifi_config_get_dtim_interval
 - WIFI STA APIs, [107](#)
- wifi_config_get_listen_interval
 - WIFI STA APIs, [108](#)
- wifi_config_get_mac_address
 - WIFI STA APIs, [108](#)
- wifi_config_get_opmode
 - WIFI STA APIs, [109](#)
- wifi_config_get_skip_dtim
 - WIFI STA APIs, [109](#)
- wifi_config_get_ssid
 - WIFI STA APIs, [109](#)
- wifi_config_set_bandwidth
 - WIFI STA APIs, [110](#)
- wifi_config_set_bssid
 - WIFI STA APIs, [110](#)
- wifi_config_set_channel
 - WIFI STA APIs, [111](#)
- wifi_config_set_dtim_interval
 - WIFI STA APIs, [111](#)
- wifi_config_set_listen_interval
 - WIFI STA APIs, [112](#)
- wifi_config_set_mac_address
 - WIFI STA APIs, [112](#)
- wifi_config_set_opmode
 - WIFI STA APIs, [113](#)
- wifi_config_set_skip_dtim
 - WIFI STA APIs, [113](#)
- wifi_config_set_ssid
 - WIFI STA APIs, [113](#)
- wifi_config_t, [212](#)
 - ap_config, [213](#)
 - sta_config, [213](#)
- wifi_connection_connect
 - WIFI STA APIs, [115](#)
- wifi_connection_disconnect_ap
 - WIFI STA APIs, [115](#)
- wifi_connection_disconnect_sta
 - WIFI STA APIs, [115](#)
- wifi_connection_get_rssi
 - WIFI STA APIs, [116](#)
- wifi_connection_register_event_handler
 - WIFI STA APIs, [116](#)
- wifi_connection_scan_start
 - WIFI STA APIs, [117](#)
- wifi_connection_unregister_event_handler

- WIFI STA APIs, 117
- wifi_deinit
 - WIFI STA APIs, 118
- wifi_event_cb_t
 - WIFI Common APIs, 96
- wifi_event_handler_t
 - WIFI STA APIs, 102
- wifi_event_info_t, 213
 - connected, 214
 - disconnected, 214
 - got_ip, 214
 - scan_done, 214
- wifi_event_loop_init
 - WIFI Common APIs, 97
- wifi_event_loop_send
 - WIFI Common APIs, 98
- wifi_event_loop_set_cb
 - WIFI Common APIs, 98
- wifi_event_notify_cb_t
 - WIFI APIs, 93
- wifi_event_process_handler
 - WIFI APIs, 94
 - WIFI Common APIs, 99
- wifi_event_sta_connected_t, 214
 - authmode, 215
 - bssid, 215
 - channel, 215
 - ssid, 215
 - ssid_len, 215
- wifi_event_sta_disconnected_t, 215
 - bssid, 216
 - reason, 216
 - ssid, 216
 - ssid_len, 216
- wifi_event_sta_got_ip_t, 216
 - ip_changed, 217
- wifi_event_sta_scan_done_t, 217
 - number, 217
 - scan_id, 217
 - status, 217
- wifi_event_t
 - Enumeration, 126
- wifi_fast_connect_get_mode
 - WIFI STA APIs, 118
- wifi_fast_connect_set_mode
 - WIFI STA APIs, 119
- wifi_fast_connect_start
 - WIFI STA APIs, 119
- wifi_fast_scan_threshold_t, 218
 - authmode, 218
 - rsi, 218
- wifi_get_config
 - WIFI STA APIs, 119
- wifi_init
 - WIFI STA APIs, 120
- wifi_init_complete_cb_t
 - WIFI STA APIs, 102
- wifi_init_config_t, 218
 - event_handler, 219
 - magic, 219
- wifi_install_default_event_handlers
 - WIFI APIs, 94
- wifi_mode_t
 - Enumeration, 127
- wifi_reason_code_t
 - Enumeration, 127
- wifi_register_event_handler
 - WIFI APIs, 94
- wifi_result_t
 - WIFI STA APIs, 103
- wifi_scan_config_t, 219
 - bssid, 220
 - channel, 220
 - scan_time, 220
 - scan_type, 220
 - show_hidden, 220
 - ssid, 220
- wifi_scan_get_ap_list
 - WIFI STA APIs, 120
- wifi_scan_get_ap_num
 - WIFI STA APIs, 121
- wifi_scan_get_ap_records
 - WIFI STA APIs, 121
- wifi_scan_info_t, 220
 - auth_mode, 221
 - beacon_interval, 221
 - bssid, 221
 - capability_info, 221
 - channel, 221
 - group_cipher, 222
 - pairwise_cipher, 222
 - rsi, 222
 - ssid, 222
 - ssid_length, 222
- wifi_scan_list_t, 222
 - ap_record, 223
 - num, 223
- wifi_scan_method_t
 - Enumeration, 128
- wifi_scan_scan_stop
 - WIFI STA APIs, 122
- wifi_scan_start
 - WIFI STA APIs, 122
- wifi_scan_time_t, 223
 - active, 223
 - passive, 223
- wifi_scan_type_t
 - Enumeration, 128
- wifi_set_config
 - WIFI STA APIs, 122
- wifi_sort_method_t
 - Enumeration, 129
- wifi_sta_config_t, 224
 - bssid, 224
 - bssid_present, 224
 - password, 224

- password_length, [225](#)
- scan_method, [225](#)
- sort_method, [225](#)
- ssid, [225](#)
- ssid_length, [225](#)
- threshold, [225](#)
- wifi_sta_get_ap_info
 - WIFI STA APIs, [123](#)
- wifi_start
 - WIFI STA APIs, [123](#)
- wifi_stop
 - WIFI STA APIs, [124](#)
- window
 - LE_GAP_SCAN_PARAM_T, [160](#)
- wpa_data
 - auto_conn_info_t, [133](#)
 - mw_wifi_auto_connect_ap_info_t, [202](#)
 - wifi_auto_connect_info_t, [212](#)
- wpa_ie
 - auto_conn_info_t, [133](#)
 - mw_wifi_auto_connect_ap_info_t, [203](#)
 - wifi_auto_connect_info_t, [212](#)