

GigaDevice Semiconductor Inc.

GD32VW553 BLE Development Guide

Application Note

AN152

Revision 1.2

(Jul.2024)

Table of Contents

Table of Contents	2
List of Figures	7
List of Tables	8
1. Overview of BLE SDK.....	9
1.1. BLE software framework.....	9
2. BLE API.....	11
2.1. BLE adapter API.....	11
2.1.1. Adapter message type.....	11
2.1.2. ble_adp_init.....	12
2.1.3. ble_adp_callback_register	13
2.1.4. ble_adp_callback_unregister	13
2.1.5. ble_adp_reset	13
2.1.6. ble_adp_disable.....	13
2.1.7. ble_adp_cfg	14
2.1.8. ble_adp_chann_map_set	14
2.1.9. ble_adp_loc_irk_set.....	15
2.1.10. ble_adp_loc_irk_get.....	15
2.1.11. ble_adp_identity_addr_get	15
2.1.12. ble_adp_name_set	15
2.1.13. ble_adp_local_ver_get.....	16
2.1.14. ble_adp_sugg_dft_data_len_get	16
2.1.15. ble_adp_tx_pwr_range_get	16
2.1.16. ble_adp_max_data_len_get	16
2.1.17. ble_adp_adv_sets_num_get.....	17
2.1.18. ble_adp_addr_resolve	17
2.1.19. ble_adp_static_random_addr_gen	17
2.1.20. ble_adp_resolvable_private_addr_gen	18
2.1.21. ble_adp_none_resolvable_private_addr_gen	18
2.1.22. ble_adp_test_tx.....	18
2.1.23. ble_adp_test_rx	19
2.1.24. ble_adp_test_end	19
2.2. BLE advertising API	20
2.2.1. Advertising message type.....	20
2.2.2. ble_adv_init.....	20
2.2.3. ble_adv_deinit.....	20
2.2.4. ble_adv_create	21
2.2.5. ble_adv_start	21

2.2.6.	ble_adv_restart	22
2.2.7.	ble_adv_stop.....	22
2.2.8.	ble_adv_remove	23
2.2.9.	ble_adv_data_update	23
2.3.	BLE advertising data API	23
2.3.1.	ble_adv_find.....	24
2.3.2.	ble_adv_cmpl_name_find	24
2.3.3.	ble_adv_short_name_find	24
2.3.4.	ble_adv_srv_uuid_find.....	25
2.3.5.	ble_adv_appearance_find	25
2.4.	BLE scan API	26
2.4.1.	Scan message type	26
2.4.2.	ble_scan_init.....	26
2.4.3.	ble_scan_reinit.....	26
2.4.4.	ble_scan_callback_register	27
2.4.5.	ble_scan_callback_unregister	27
2.4.6.	ble_scan_enable.....	27
2.4.7.	ble_scan_disable	27
2.4.8.	ble_scan_param_set	28
2.5.	BLE connection API	28
2.5.1.	Connection message type	28
2.5.2.	ble_conn_init.....	33
2.5.3.	ble_conn_callback_register	33
2.5.4.	ble_conn_callback_unregister	33
2.5.5.	ble_conn_connect.....	34
2.5.6.	ble_conn_disconnect.....	34
2.5.7.	ble_conn_connect_cancel	35
2.5.8.	ble_conn_sec_info_set	35
2.5.9.	ble_conn_peer_name_get	36
2.5.10.	ble_conn_peer_feats_get	36
2.5.11.	ble_conn_peer_appearance_get.....	36
2.5.12.	ble_conn_peer_version_get	37
2.5.13.	ble_conn_peer_slave_prefer_param_get.....	37
2.5.14.	ble_conn_peer_addr_resolution_support_get.....	38
2.5.15.	ble_conn_peer_rpa_only_get	38
2.5.16.	ble_conn_peer_db_hash_get	38
2.5.17.	ble_conn_phy_get	39
2.5.18.	ble_conn_phy_set.....	39
2.5.19.	ble_conn_pkt_size_set	40
2.5.20.	ble_conn_chann_map_get	40
2.5.21.	ble_conn_ping_to_get	41
2.5.22.	ble_conn_ping_to_set.....	41

2.5.23.	ble_conn_rssi_get.....	41
2.5.24.	ble_conn_param_update_req.....	42
2.5.25.	ble_conn_per_adv_sync_trans.....	42
2.5.26.	ble_conn_name_get_cfm	43
2.5.27.	ble_conn_appearance_get_cfm	44
2.5.28.	ble_conn_slave_prefer_param_get_cfm	44
2.5.29.	ble_conn_name_set_cfm.....	45
2.5.30.	ble_conn_appearance_set_cfm.....	45
2.5.31.	ble_conn_param_update_cfm	46
2.5.32.	ble_conn_local_tx_pwr_get	46
2.5.33.	ble_conn_peer_tx_pwr_get	47
2.5.34.	ble_conn_tx_pwr_report_ctrl	47
2.5.35.	ble_conn_path_loss_ctrl	48
2.5.36.	ble_conn_enable_central_feat.....	49
2.6.	BLE security API.....	49
2.6.1.	Security message type	49
2.6.2.	ble_sec_init.....	51
2.6.3.	ble_sec_callback_register	51
2.6.4.	ble_sec_callback_unregister	52
2.6.5.	ble_sec_security_req.....	52
2.6.6.	ble_sec_bond_req	52
2.6.7.	ble_sec_encrypt_req	53
2.6.8.	ble_sec_key_press_notify	53
2.6.9.	ble_sec_key_display_enter_cfm.....	54
2.6.10.	ble_sec_oob_req_cfm	54
2.6.11.	ble_sec_nc_cfm	54
2.6.12.	ble_sec_ltk_req_cfm.....	55
2.6.13.	ble_sec_irk_req_cfm.....	55
2.6.14.	ble_sec_csrk_req_cfm.....	56
2.6.15.	ble_sec_encrypt_req_cfm.....	56
2.6.16.	ble_sec_pairing_req_cfm.....	56
2.6.17.	ble_sec_oob_data_req_cfm	57
2.6.18.	ble_sec_oob_data_gen	57
2.7.	BLE list API	58
2.7.1.	List message type.....	58
2.7.2.	ble_list_init	58
2.7.3.	ble_list_callback_register.....	59
2.7.4.	ble_list_callback_unregister.....	59
2.7.5.	ble_fal_op	59
2.7.6.	ble_fal_list_set	59
2.7.7.	ble_fal_clear.....	60
2.7.8.	ble_fal_size_get.....	60
2.7.9.	ble_ral_op	60

2.7.10.	ble_ral_list_set.....	61
2.7.11.	ble_ral_clear.....	61
2.7.12.	ble_ral_size_get.....	62
2.7.13.	ble_loc_rpa_get.....	62
2.7.14.	ble_peer_rpa_get.....	62
2.7.15.	ble_pal_op.....	63
2.7.16.	ble_pal_list_set.....	63
2.7.17.	ble_pal_clear.....	63
2.7.18.	ble_pal_size_get.....	64
2.8.	BLE periodic sync API.....	64
2.8.1.	Periodic sync message type.....	64
2.8.2.	ble_per_sync_init.....	65
2.8.3.	ble_per_sync_callback_register.....	65
2.8.4.	ble_per_sync_callback_unregister.....	65
2.8.5.	ble_per_sync_start.....	65
2.8.6.	ble_per_sync_cancel.....	66
2.8.7.	ble_per_sync_terminate.....	66
2.8.8.	ble_per_sync_ctrl.....	67
2.9.	BLE storage API.....	67
2.9.1.	ble_storage_init.....	67
2.9.2.	ble_peer_data_bond_store.....	67
2.9.3.	ble_peer_data_bond_load.....	68
2.9.4.	ble_peer_data_delete.....	68
2.9.5.	ble_peer_all_addr_get.....	69
2.9.6.	ble_svc_data_save.....	69
2.9.7.	ble_svc_data_load.....	70
2.10.	BLE gatts API.....	70
2.10.1.	gatts message type.....	70
2.10.2.	ble_gatts_init.....	72
2.10.3.	ble_gatts_svc_add.....	72
2.10.4.	ble_gatts_svc_rmv.....	73
2.10.5.	ble_gatts_ntf_ind_send.....	73
2.10.6.	ble_gatts_ntf_ind_send_by_handle.....	73
2.10.7.	ble_gatts_ntf_ind_mtp_send.....	74
2.10.8.	ble_gatts_mtu_get.....	75
2.10.9.	ble_gatts_svc_attr_write_cfm.....	75
2.10.10.	ble_gatts_svc_attr_read_cfm.....	75
2.10.11.	ble_gatts_get_start_hdl.....	76
2.11.	BLE gattc API.....	76
2.11.1.	gattc message type.....	76
2.11.2.	ble_gattc_init.....	77
2.11.3.	ble_gattc_start_discovery.....	77

2.11.4.	ble_gattc_svc_reg.....	78
2.11.5.	ble_gattc_svc_unreg.....	78
2.11.6.	ble_gattc_read.....	78
2.11.7.	ble_gattc_write_req.....	79
2.11.8.	ble_gattc_write_cmd.....	79
2.11.9.	ble_gattc_write_signed.....	80
2.11.10.	ble_gattc_mtu_update.....	80
2.11.11.	ble_gattc_mtu_get.....	80
2.11.12.	ble_gattc_find_char_handle.....	81
2.11.13.	ble_gattc_find_desc_handle.....	81
2.12.	BLE export API.....	81
2.12.1.	ble_stack_init.....	81
2.12.2.	ble_stack_task_resume.....	82
2.12.3.	ble_stack_task_init.....	82
2.12.4.	ble_stack_task_deinit.....	82
2.12.5.	ble_app_task_init.....	83
2.12.6.	ble_app_task_deinit.....	83
2.12.7.	ble_local_app_msg_send.....	83
2.12.8.	ble_app_msg_hdl_reg.....	83
2.12.9.	ble_sleep_mode_set.....	84
2.12.10.	ble_sleep_mode_get.....	84
2.12.11.	ble_core_is_deep_sleep.....	84
2.12.12.	ble_modem_config.....	84
2.12.13.	ble_work_status_set.....	85
2.12.14.	ble_work_status_get.....	85
2.12.15.	ble_internal_encode.....	85
2.12.16.	ble_internal_decode.....	85
2.12.17.	ble_register_hci_uart.....	86
3.	Application examples.....	87
3.1.	Scan.....	87
3.2.	Advertising.....	88
3.3.	GATT server application.....	93
3.3.1.	Adding a service.....	93
3.3.2.	Service attribute database.....	93
3.3.3.	Service attribute read and write.....	95
3.4.	BLE distribution network.....	97
3.4.1.	Process of Blue courier.....	97
3.4.2.	GATT description.....	98
3.4.3.	Advertising data.....	99
3.4.4.	Frame format.....	99
4.	Revision history.....	103

List of Figures

Figure 1-1. BLE software framework	9
Figure 3-1. Process of Blue courier	98

List of Tables

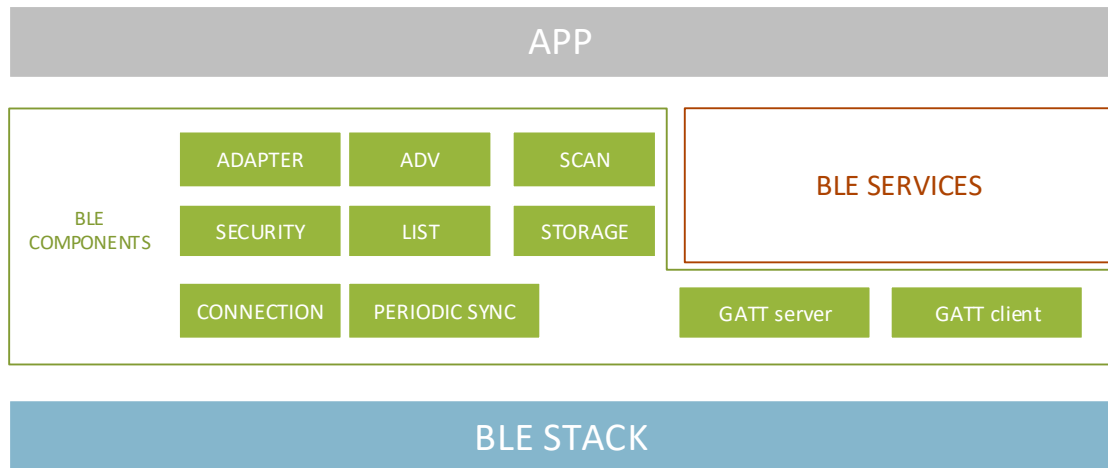
Table 3-1. Example code of scan event handler	87
Table 3-2. Example code of configure scan parameters	88
Table 3-3. Example code of enable scan	88
Table 3-4. Example code of advertising event handler	89
Table 3-5. Example code of create advertising	90
Table 3-6. Example code of enable advertising	92
Table 3-7. Example code of add a service	93
Table 3-8. Example code of service database	93
Table 3-9. Example code of attribute read and write function	95
Table 3-10. Example code of send notification	97
Table 3-11. Distribution network service UUID	98
Table 3-12. Service UUID in advertising data	99
Table 3-13. Frame format of blue courier	99
Table 3-14. Frame control field	99
Table 3-15. Content of management frame	100
Table 3-16. Content of data frame	101
Table 4-1. Revision history	103

1. Overview of BLE SDK

The GD32VW553 series chip is a 32-bit microcontroller (MCU) with RISC-V as the core, which contains Wi-Fi4/Wi-Fi6 and BLE5.3 connection technologies. GD32VW553 Wi-Fi+BLE SDK integrates the Wi-Fi driver, BLE driver, LwIP TCP/IP protocol stack, MbedTLS, and other components, allowing developers to quickly develop IoT applications based on GD32VW553. This document describes the BLE software framework and related API interfaces aiming to help developers become familiar with BLE APIs and use them to develop their own applications. For related Wi-Fi information, please refer to the "AN158 GD32VW553 Wi-Fi Development Guide".

1.1. BLE software framework

Figure 1-1. BLE software framework



As shown in [Figure 1-1. BLE software framework](#), the GD32VW553 BLE software part consists of four modules: BLE STACK, BLE COMPONENTS, BLE services, and BLE APP.

BLE STACK is the implementation of the BLE protocol stack, which includes GAP, GATT, SMP, L2CAP, HCI, LL, and other modules. BLE STACK runs in a separate task and interacts with BLE COMPONENTS through TASK messages. APP needs to operate STACK through BLE COMPONENTS.

BLE COMPONENTS consists of multiple components, and runs in the same task as BLE service and BLE APP to provide APP with interfaces for STACK control and status notification, etc. Note that most operations of BLE are executed asynchronously. APP needs to register a callback handler in each module, and BLE COMPONENTS will notify APP of the API call execution result or report the operation request initiated by the peer device in the callback function. Each component is independent of each other. APP can select different components to initialize them and register the corresponding callback functions as required.

The BLE ADAPTER module mainly provides interfaces for configuring and obtaining local BLE related attributes. [BLE adapter API](#) introduces how to use API of the ADAPTER module.

The BLE ADV module mainly provides interfaces for creating/deleting advertising sets, starting/stopping sending advertising packets, etc. [BLE advertising API](#) introduces how to use API of the ADV module, and [BLE advertising data API](#) provides some interfaces for searching for specific AD type data in advertising data.

The BLE SCAN module mainly provides interfaces for searching for advertising sets and reports the search results to the APP. [BLE scan API](#) introduces how to use API of the SCAN module.

The BLE CONNECTION module mainly provides interfaces for creating connections, obtaining peer device information, and obtaining or setting connection parameters, etc. [BLE connection API](#) introduces how to use API of the CONNECTION module.

The BLE SECURITY module mainly provides interfaces required for interaction during pairing, authentication, encryption, and other processes. [BLE security API](#) introduces how to use API of the SECURITY module.

The BLE LIST module mainly provides interfaces for operating FAL, RAL, and PAL, including operations such as adding devices to the list, deleting devices from the list, and clearing the list. [BLE list API](#) introduces how to use API of the LIST module .

The BLE PERIODIC SYNC module mainly provides interfaces for synchronizing periodic advertising, reporting received periodic advertising data, etc. [BLE periodic sync API](#) introduces how to use API of the PERIODIC SYNC module .

The BLE STORAGE module uses flash to store and manage the bond information of the peer device. The bond information includes peer_irk, peer_ltk, peer_csrk, local_irk, local_ltk, local_csrk, etc. [BLE storage API](#) introduces how to use API of the STORAGE module.

The BLE GATT server module mainly provides interfaces for registering/deleting GATT service, sending notification/indication to GATT client, etc. [BLE gatts API](#) introduces how to use API of the GATT server module.

The BLE GATT client module mainly provides the following functions: initiate GATT discovery, read and write attribute in the peer GATT server. [BLE gattc API](#) introduces API usage of GATT client module.

BLE services are different services and profiles realized based on GATT server and GATT client modules, including BAS and DIS, etc. Users can also realize private services by using GATT server and GATT client interfaces required.

The BLE APP layer is a collection of multiple applications, such as blue courier (Bluetooth distribution network) and user-defined applications. APP can register callback functions with different modules to process corresponding messages according to different requirements.

2. BLE API

2.1. BLE adapter API

The header file is `ble_adapter.h`.

The BLE adapter module mainly provides interfaces for configuring and obtaining local BLE related settings.

2.1.1. Adapter message type

APP can register a callback function in the BLE adapter module, and the BLE adapter module will send the following event message to APP through the callback function.

- `BLE_ADP_EVT_ENABLE_CMPL_INFO`

This message will be sent after the BLE adapter is initialized. The message data type is `ble_adp_info_t`, including whether the initialization is successful; if yes, local attributes such as local version and local IRK will also be reported.

APP can only perform BLE related operations after it receives this message and the status indicates that the initialization is successful.

- `BLE_ADP_EVT_RESET_CMPL_INFO`

This message will be sent after the BLE adapter is reset. The message data type is `uint16_t`, indicating whether the reset is successful.

- `BLE_ADP_EVT_DISABLE_CMPL_INFO`

This message returns the result of APP calling `ble_adp_disable` API to disable the BLE module. The message data type is `uint16_t`, indicating whether the disable is successful.

- `BLE_ADP_EVT_CHANN_MAP_SET_RSP`

This message returns the result of APP calling `ble_adp_chann_map_set` API to set the channel map. The message data type is `uint16_t`, indicating whether the channel map is set successfully.

- `BLE_ADP_EVT_LOC_IRK_SET_RSP`

This message returns the result of APP calling `ble_adp_loc_irk_set` API to set the local IRK. The message data type is `uint16_t`, indicating whether the local IRK is set successfully.

- `BLE_ADP_EVT_LOC_ADDR_INFO`

This message is used to notify APP of new address information after the local address changes, for example, after RPA timeout. The message data type is `ble_gap_local_addr_info_t`.

- BLE_ADP_EVT_NAME_SET_RSP

This message returns the result of APP calling ble_adp_name_set API to set the local name. The message data type is uint16_t, and the status indicates whether the local name is set successfully.

- BLE_ADP_EVT_ADDR_RESLV_RSP

This message returns the result of APP calling ble_adp_addr_resolve API to resolve the passed in RPA. The message data type is ble_gap_addr_resolve_rsp_t, including whether the RPA is resolved successful; if yes, it also contains the address after the resolving and the corresponding IRK information.

- BLE_ADP_EVT_RAND_ADDR_GEN_RSP

This message returns the result of APP calling ble_adp_none_resolvable_private_addr_gen API, ble_adp_static_random_addr_gen API, or ble_adp_resolvable_private_addr_gen API to generate a random address. The message data type is ble_gap_rand_addr_gen_rsp_t. If the random address is successfully generated, the corresponding address information is also provided.

- BLE_ADP_EVT_TEST_TX_RSP

This message returns the result of APP calling ble_adp_test_tx API. The message data type is uint16_t, indicating whether the tx test starts to be executed successfully.

- BLE_ADP_EVT_TEST_RX_RSP

This message returns the result of APP calling ble_adp_test_rx API. The message data type is uint16_t, indicating whether the rx test starts to be executed successfully.

- BLE_ADP_EVT_TEST_END_RSP

This message returns the result of APP calling ble_adp_test_end API. The message data type is ble_gap_test_end_rsp_t, including whether the test is ended successfully.

- BLE_ADP_EVT_TEST_RX_PKT_INFO

This message is used to notify APP about the successfully received packet number after test rx mode is ended. The message data type is ble_gap_test_rx_pkt_info_t.

2.1.2. ble_adp_init

Prototype: ble_status_t ble_adp_init(void)

Function: Initialize the BLE adapter module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.3. **ble_adp_callback_register**

Prototype: ble_status_t ble_adp_callback_register(ble_adp_evt_handler_t callback)

Function: Register the callback function that processes BLE adapter messages. For the description of adapter messages, see [Adapter message type](#).

Input parameter: callback, callback function pointer

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.4. **ble_adp_callback_unregister**

Prototype: ble_status_t ble_adp_callback_unregister(ble_adp_evt_handler_t callback)

Function: Unregister the callback function from BLE adapter module

Input parameter: callback, callback function to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.5. **ble_adp_reset**

Prototype: ble_status_t ble_adp_reset(void)

Function: Reset BLE protocol stack and each module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the resetting, a BLE_ADP_EVT_RESET_CMPL_INFO message is sent to the callback function

2.1.6. **ble_adp_disable**

Prototype: ble_status_t ble_adp_disable(void)

Function: Disable BLE protocol stack and each module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the disabling, a BLE_ADP_EVT_DISABLE_CMPL_INFO message is sent to the callback function

2.1.7. ble_adp_cfg

Prototype: ble_status_t ble_adp_cfg(ble_adp_config_t *p_adp_config)

Function: Configure BLE adapter

Input parameter: p_adp_config, adapter config structure pointer, which can be used

to configure the role, privacy, and other attributes of the device

If keys_user_mgr in config is set to true, APP is required to save

and manage keys. APP can call the storage API provided

in [BLE storage API](#) to access keys or manage them in the way

it needs. Otherwise, keys are managed by the ble security module.

If the APP does not need to manage relevant information,

it can call ble_peer_data_bond_load to get the saved key information.

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the configuration, a BLE_ADP_EVT_ENABLE_CMPL_INFO message

is sent to the callback function

2.1.8. ble_adp_chann_map_set

Prototype: ble_status_t ble_adp_chann_map_set(uint8_t *p_chann_map)

Function: Set the channel map available for BLE

Input parameter: p_chann_map, channel map array, whose length is 5

bytes and effective bits are the lower 37 bits. Bit 0 of byte 0 is set to use

channel index 0, bit 1 of byte 0 is set to use channel index 1, and so on

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the setting, a BLE_ADP_EVT_CHANN_MAP_SET_RSP message is

sent to the callback function

2.1.9. ble_adp_loc_irk_set

Prototype: ble_status_t ble_adp_loc_irk_set(uint8_t *p_irk)

Function: Set local IRK

Input parameter: p_irk, the IRK pointer to be set, whose length is 16 bytes

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the setting, a BLE_ADP_EVT_LOC_IRK_SET_RSP message is sent to the callback function

2.1.10. ble_adp_loc_irk_get

Prototype: ble_status_t ble_adp_loc_irk_get (uint8_t *p_irk)

Function: Get local IRK used by BLE adapter

Input parameter: None

Output parameter: p_irk, local IRK pointer, whose length is 16 bytes, is used to save the obtained local IRK information

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.11. ble_adp_identity_addr_get

Prototype: ble_status_t ble_adp_identity_addr_get (ble_gap_addr_t *p_id_addr)

Function: Get identity address used by BLE adapter

Input parameter: None

Output parameter: p_id_addr, identity address pointer, including address type and address value

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.12. ble_adp_name_set

Prototype: ble_status_t ble_adp_name_set (uint8_t *p_name, uint8_t name_len)

Function: Set device name used by BLE adapter

Input parameter: p_name, device name pointer

name_len, device name length

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

After the setting, a `BLE_ADP_EVT_NAME_SET_RSP` message is sent to the callback function

2.1.13. `ble_adp_local_ver_get`

Prototype: `ble_status_t ble_adp_local_ver_get(ble_gap_local_ver_t *p_val)`

Function: Get BLE adapter version information

Input parameter: None

Output parameter: `p_val`, local version structure pointer, including hci version, Imp version, etc.

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.1.14. `ble_adp_sugg_dft_data_len_get`

Prototype: `ble_status_t ble_adp_sugg_dft_data_len_get(ble_gap_sugg_dft_data_t *p_data)`

Function: Get default transmit data parameters of BLE adapter

Input parameter: None

Output parameter: `p_data`, suggest data structure pointer, including max tx time and max tx octets

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.1.15. `ble_adp_tx_pwr_range_get`

Prototype: `ble_status_t ble_adp_tx_pwr_range_get(ble_gap_tx_pwr_range_t *p_val)`

Function: Get the BLE adapter transmit power range

Input parameter: None

Output parameter: `p_val`, tx power range structure pointer, including min tx power and max tx power

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.1.16. `ble_adp_max_data_len_get`

Prototype: `ble_status_t ble_adp_max_data_len_get(ble_gap_max_data_len_t *p_len)`

Function: Get BLE adapter max data length information

Input parameter: None

Output parameter: p_len, max data length structure pointer, including max tx octets, max tx time, max rx octets, and max rx time

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.17. ble_adp_adv_sets_num_get

Prototype: ble_status_t ble_adp_adv_sets_num_get (uint8_t *p_val)

Function: Get the maximum number of advertising sets supported by BLE adapter

Input parameter: None

Output parameter: p_val, advertising set number pointer

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.1.18. ble_adp_addr_resolve

Prototype: ble_status_t ble_adp_addr_resolve(uint8_t *p_addr, uint8_t *p_irk, uint8_t irk_num)

Function: Use the keys in the provided IRK list in turn to resolve the input RPA

Input parameter: p_addr, resolvable private address to be resolved

p_irk, IRK list pointer

irk_num, the number of keys in the IRK list

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined

in ble_status_t on failure After execution, a

BLE_ADP_EVT_ADDR_RESLV_RSP message is sent to

the callback function. If the provided address can be resolved,

the message data includes the resolved identity address and the used IRK.

2.1.19. ble_adp_static_random_addr_gen

Prototype: ble_status_t ble_adp_static_random_addr_gen(void)

Function: Generate static random address

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_ADP_EVT_RAND_ADDR_GEN_RSP` message is sent to the callback function.

2.1.20. **ble_adp_resolvable_private_addr_gen**

Prototype: `ble_status_t ble_adp_resolvable_private_addr_gen(void)`

Function: Generate static resolvable private address

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_ADP_EVT_RAND_ADDR_GEN_RSP` message is sent to the callback function.

2.1.21. **ble_adp_none_resolvable_private_addr_gen**

Prototype: `ble_status_t ble_adp_none_resolvable_private_addr_gen(void)`

Function: Generate static non-resolvable private address

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_ADP_EVT_RAND_ADDR_GEN_RSP` message is sent to the callback function.

2.1.22. **ble_adp_test_tx**

Prototype: `ble_status_t ble_adp_test_tx(uint8_t chann, uint8_t tx_data_len, uint8_t tx_pkt_payload, uint8_t phy, int8_t tx_pwr_lvl)`

Function: Configure BLE controller to enter the test mode and send test packet

Input parameter: `chann`, tx rf channel index, whose range is 0x00-0x27

`tx_data_len`, length of tx packet, whose range is 0x00-0xFF

`tx_pkt_payload`, type of tx packet, whose range is 0x00-0x07

`phy`, PHY used by tx, 1: 1M, 2: 2M, 3: coded S=8, 4: coded S=2

`tx_pwr_lvl`: tx power

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined

in `ble_status_t` on failure After execution, a `BLE_ADP_EVT_TEST_TX_RSP` message is sent to the callback function

2.1.23. `ble_adp_test_rx`

Prototype: `ble_status_t ble_adp_test_rx(uint8_t chann, uint8_t phy, uint8_t modulation_idx)`

Function: Configure BLE controller to enter the test mode and receive test packet

Input parameter: `chann`, rf channel index used by rx, whose range is 0x00-0x27

`phy`, PHY used by rx, 1: 1M, 2: 2M, 3: coded

`modulation_idx`: Whether BLE controller has stable modulation index

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined

in `ble_status_t` on failure After execution, a `BLE_ADP_EVT_TEST_RX_RSP` message is sent to the callback function

2.1.24. `ble_adp_test_end`

Prototype: `ble_status_t ble_adp_test_end(void)`

Function: Configure BLE controller to exit the test mode

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined

in `ble_status_t` on failure After execution, a `BLE_ADP_EVT_TEST_END_RSP` message is sent to the callback function. If test rx mode is ended, a `BLE_ADP_EVT_TEST_RX_PKT_INFO` message is also sent.

2.2. BLE advertising API

The header file is `ble_adv.h`.

The BLE advertising module mainly provides interfaces for creating/deleting advertising sets, starting/stopping sending advertising packets, etc.

2.2.1. Advertising message type

■ BLE_ADV_EVT_STATE_CHG

This message is used to notify APP after the state of advertising sets changes. The state of advertising sets is defined as `ble_adv_state_t`, including the new state, the reason for state change, and the changed adv index.

■ BLE_ADV_EVT_DATA_UPDATE_RSP

This message is a response to APP calling `ble_adv_data_update` to update the data of the advertising set being used. The message data type is `ble_adv_data_update_rsp_t`, including the updated advertising data type and the update success or failure state.

■ BLE_ADV_EVT_SCAN_REQ_RCV

If scan request notification is enabled upon the creation of advertising set, and a scan request packet is received after advertising is enabled, APP will receive this message. The message data type is `ble_adv_scan_req_rcv_t`, including the set address for sending the scan request.

2.2.2. `ble_adv_init`

Prototype: `ble_status_t ble_adv_init(void)`

Function: Initialize the BLE adv module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.2.3. `ble_adv_deinit`

Prototype: `ble_status_t ble_adv_deinit(void)`

Function: Release the BLE adv module and used resources

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.2.4. **ble_adv_create**

Prototype: ble_status_t ble_adv_create(ble_adv_param_t *p_param,
ble_adv_evt_handler_t hdlr, void *p_context)

Function: Create BLE advertising set

Input parameter: p_param, advertising parameter structure pointer, which can be used
to configure adv type, interval, phy, and other parameters
hdlr, a handler that registers messages related to the adv.

For the description of adv messages, see [Advertising message type](#).

p_context, a parameter that can be additionally returned to
the message handler

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After the advertising set is successfully created, a BLE_ADV_EVT_STATE_CHG message is sent to the registered message handler, and the state is BLE_ADV_STATE_CREATE. In addition, adv index can be obtained from the message and used in subsequent APIs.

2.2.5. **ble_adv_start**

Prototype: ble_status_t ble_adv_start(uint8_t adv_idx, ble_adv_data_set_t *p_adv_data,
ble_adv_data_set_t *p_scan_rsp_data, ble_adv_data_set_t *p_per_adv_data)

Function: Set advertising set data and start sending advertising packet

Input parameter: adv_idx, advertising index

p_adv_data, advertising data structure pointer, data can be generated by
the ble adv module through configuration or directly set by the
caller

p_scan_rsp_data, scan response which needs to be set when the created
advertising set is scannable advertising

p_per_adv_data, periodic advertising data structure pointer, which needs to
be set when the created advertising set is periodic advertising

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in

ble_status_t on failure. After the function is called,
a BLE_ADV_EVT_STATE_CHG message is sent to the message handler registered when advertising is created. According to the advertising data need to set, there may be messages whose state is
BLE_ADV_STATE_ADV_DATA_SET,
BLE_ADV_STATE_SCAN_RSP_DATA_SET,
or BLE_ADV_STATE_PER_ADV_DATA_SET.
Finally, there is a message whose state is BLE_ADV_STATE_START

2.2.6. ble_adv_restart

Prototype: ble_status_t ble_adv_restart(uint8_t adv_idx)

Function: Resend advertising packet after the advertising set is stopped

Input parameter: adv_idx, advertising index

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined
in ble_status_t on failure. After the advertising is restarted successfully,
a BLE_ADV_EVT_STATE_CHG message is sent to the message handler registered when ble_adv_create API is called, and the
state is BLE_ADV_STATE_START

2.2.7. ble_adv_stop

Prototype: ble_status_t ble_adv_stop(uint8_t adv_idx)

Function: Stop sending advertising packets

Input parameter: adv_idx, advertising index

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined
in ble_status_t on failure. After the advertising set stops to be sent,
a BLE_ADV_EVT_STATE_CHG message is sent to the message handler registered when ble_adv_create API is called, and the
state is BLE_ADV_STATE_CREATE

2.2.8. **ble_adv_remove**

Prototype: `ble_status_t ble_adv_remove(uint8_t adv_idx)`

Function: Delete the advertising set that no longer sends advertising packets.

If the advertising set is sending advertising packets, that is, the state is `BLE_ADV_STATE_START`, first call `ble_adv_stop` to stop it, and then call this function to remove it.

Input parameter: `adv_idx`, advertising index

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.2.9. **ble_adv_data_update**

Prototype: `ble_status_t ble_adv_data_update(uint8_t adv_idx, ble_adv_data_set_t *p_adv_data, ble_adv_data_set_t *p_scan_rsp_data, ble_adv_data_set_t *p_per_adv_data)`

Function: Update the adv data, scan response data, and periodic adv data of the advertising set which is sending advertising packets and whose state is `BLE_ADV_STATE_START`

Input parameters: `adv_idx`, advertising index

`p_adv_data`, advertising data structure pointer

`p_scan_rsp_data`, scan response data structure pointer

`p_per_adv_data`, periodic advertising data structure pointer

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

After execution, a `BLE_ADV_EVT_DATA_UPDATE_RSP`

message is sent to the callback function

2.3. **BLE advertising data API**

The header file is `ble_adv_data.h`.

The BLE advertising data module mainly provides interfaces for searching for the specified ad type in advertising data.

2.3.1. **ble_adv_find**

Prototype: `uint8_t *ble_adv_find(uint8_t *p_data, uint16_t data_len, uint8_t ad_type, uint8_t *p_len)`

Function: Search for data of the specified ad type in advertising data

Input parameter: `p_data`, the address of advertising data for searching

`data_len`, the length of advertising data for searching

`ad_type`, the ad type to be searched

Output parameter: `p_len`, the length of the searched data value of the corresponding type

Return value: The address of the searched data value of the corresponding type.

If not found, return NULL

2.3.2. **ble_adv_cmpl_name_find**

Prototype: `bool ble_adv_cmpl_name_find(uint8_t *p_data, uint16_t data_len, uint8_t *p_name, uint16_t name_len)`

Function: Search for the specified complete name in advertising data

Input parameter: `p_data`, the address of advertising data for searching

`data_len`, the length of advertising data for searching

`p_name`, the address of the complete name to be searched

`name_len`, the length of the complete name to be searched

Output parameter: None

Return value: Return true if the specified complete name can be found in

advertising data; otherwise, return false

2.3.3. **ble_adv_short_name_find**

Prototype: `bool ble_adv_short_name_find (uint8_t *p_data, uint16_t data_len, uint8_t *p_name, uint16_t name_len_min)`

Function: Search for the specified short name in advertising data

Input parameter: `p_data`, the address of advertising data for searching

data_len, the length of advertising data for searching

p_name, the address of the short name to be searched

name_len_min, the minimum length that the short name needs to match

Output parameter: None

Return value: Return true if the specified short name can be found
in advertising data; otherwise, return false

2.3.4. ble_adv_srv_uuid_find

Prototype: bool ble_adv_srv_uuid_find(uint8_t *p_data, uint16_t data_len, ble_uuid_t *p_uuid)

Function: Search for the specified service uuid in advertising data

Input parameter: p_data, the address of advertising data for searching

data_len, the length of advertising data for searching

p_uuid, the uuid structure pointer to be searched, including uuid
length and uuid content

Output parameter: None

Return value: Return true if the specified service uuid can be found
in advertising data; otherwise, return false

2.3.5. ble_adv_appearance_find

Prototype: bool ble_adv_appearance_find(uint8_t *p_data, uint16_t data_len,
uint16_t appearance)

Function: Search for the specified appearance in advertising data

Input parameter: p_data, the address of advertising data for searching

data_len, the length of advertising data for searching

appearance, the appearance value to be searched

Output parameter: None

Return value: Return true if the specified appearance can be found
in advertising data; otherwise, return false

2.4. BLE scan API

The header file is `ble_scan.h`.

The BLE scan module mainly provides interfaces for searching for advertising data and reports the search results.

2.4.1. Scan message type

APP can register a callback function in the BLE scan module, and the BLE scan module will send the following event message to APP through the callback function.

- `BLE_SCAN_EVT_STATE_CHG`

This message is sent to the callback function when the scan state changes. The message data type is `ble_scan_state_chg_t`, including the current scan state and the reason for change.

- `BLE_SCAN_EVT_ADV_RPT`

This message is used to notify APP of the data received after the advertising packet is scanned. The message data type is `ble_gap_adv_report_info_t`. The structure contains the received advertising packet type, advertiser address, advertising sid, data, etc.

2.4.2. `ble_scan_init`

Prototype: `ble_status_t ble_scan_init(ble_gap_local_addr_type_t own_addr_type)`

Function: Initialize the BLE scan module

Input parameter: `own_addr_type`, the local address type used in the scan process

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.4.3. `ble_scan_reinit`

Prototype: `ble_status_t ble_scan_reinit(ble_gap_local_addr_type_t own_addr_type)`

Function: Reinitialize the BLE scan module

Input parameter: `own_addr_type`, the local address type used in the scan process

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.4.4. **ble_scan_callback_register**

Prototype: `ble_status_t ble_scan_callback_register(ble_scan_evt_handler_t callback)`

Function: Register the callback function for processing BLE scan messages

Input parameter: `callback`, a function that processes BLE scan messages.

For the description of scan messages, see [Scan message type](#).

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.4.5. **ble_scan_callback_unregister**

Prototype: `ble_status_t ble_scan_callback_unregister(ble_scan_evt_handler_t callback)`

Function: Unregister the callback function from BLE scan module

Input parameter: `callback`, callback function to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.4.6. **ble_scan_enable**

Prototype: `ble_status_t ble_scan_enable(void)`

Function: Enable BLE scan, and a `BLE_SCAN_EVT_ADV_RPT` message is sent to the callback function to notify it of the scanned device.

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After the enabling, a `BLE_SCAN_EVT_STATE_CHG` message is sent to the callback function, and the state is `BLE_SCAN_STATE_ENABLED`

2.4.7. **ble_scan_disable**

Prototype: `ble_status_t ble_scan_disable(void)`

Function: Disable BLE scan

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After the disabling, a `BLE_SCAN_EVT_STATE_CHG` message is sent to the callback function, and the state is `BLE_SCAN_STATE_DISABLED`

2.4.8. **ble_scan_param_set**

Prototype: `ble_status_t ble_scan_param_set (ble_gap_scan_param_t *p_param)`

Function: Set BLE scan parameters

Input parameter: `p_param`, scan parameter structure pointer, including scan type, interval, window, etc.

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.5. **BLE connection API**

The header file is `ble_conn.h`.

The BLE connection module mainly provides interfaces for creating connections, obtaining peer device information, and obtaining or setting connection parameters.

2.5.1. **Connection message type**

APP can register a callback function in the BLE connection module, and the BLE connection module will send the following event messages to APP through the callback function.

- `BLE_CONN_EVT_INIT_STATE_CHG`

This message is sent to the callback function when the state changes during active creation of connections. The data type is `ble_init_state_chg_t`, including the current state, the reason for state change, and whether the filter accept list is used.

- `BLE_CONN_EVT_STATE_CHG`

This message is sent to the callback function after the connection state changes. The data type is `ble_conn_state_chg_t`, which contains the new state. When the state is `BLE_CONN_STATE_CONNECTED`, it also contains information of connections whose structure is `ble_gap_conn_info_t`. When the state is `BLE_CONN_STATE_DISCONNECTD`, it also contains the information of disconnections whose structure is `ble_gap_disconn_info_t`.

- BLE_CONN_EVT_DISCONN_FAIL

This message is sent to the callback function when the active disconnection fails. The data type is `ble_conn_disconn_fail_t`, including the reason for disconnection failure, etc.

- BLE_CONN_EVT_PEER_NAME_GET_RSP

This message returns the result of APP calling `ble_conn_peer_name_get` to get the name information in the peer GATT database. The message data type is `ble_gap_peer_name_get_rsp_t`, including the status of the obtained attribute. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, name length, name content, etc.

- BLE_CONN_EVT_PEER_VERSION_GET_RSP

This message returns the result of APP calling `ble_conn_peer_version_get` to get the peer version information. The message data type is `ble_gap_peer_ver_get_rsp_t`, including the status of the obtained version. If the status is `BLE_ERR_NO_ERROR`, it also contains the company id, lmp version, lmp subversion, etc.

- BLE_CONN_EVT_PEER_FEATS_GET_RSP

This message returns the result of APP calling `ble_conn_peer_feats_get` to get the information of features supported by the peer device. The message data type is `ble_gap_peer_feats_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the feature array supported by the peer, etc.

- BLE_CONN_EVT_PEER_APPEARANCE_GET_RSP

This message returns the result of APP calling `ble_conn_peer_appearance_get` to get the appearance information in the peer GATT database. The message data type is `ble_gap_peer_appearance_get_rsp_t`, including the status of the obtained attribute. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, appearance, etc.

- BLE_CONN_EVT_PEER_SLV_PRF_PARAM_GET_RSP

This message returns the result of APP calling `ble_conn_peer_slave_prefer_param_get` to get the information of the attribute slave preferred parameter in the peer GATT database. The message data type is `ble_gap_slave_prefer_param_get_rsp_t`, including the status of the obtained attribute. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, slave preferred connection interval, latency, etc.

- BLE_CONN_EVT_PEER_ADDR_RESLV_GET_RSP

This message returns the result of APP calling `ble_conn_peer_addr_resolution_support_get` to get the information of the attribute central address resolution support in the peer GATT database. The message data type is `ble_gap_peer_addr_resol_get_rsp_t`, including the status of the obtained attribute. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, central address resolution support, etc.

- BLE_CONN_EVT_PEER_RPA_ONLY_GET_RSP

This message returns the result of APP calling `ble_conn_peer_rpa_only_get` to get the information of the attribute resolvable private address only in the peer GATT database. The message data type is `ble_gap_peer_rpa_only_get_rsp_t`, including the status of the obtained attribute. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, resolvable private address only, etc.

■ `BLE_CONN_EVT_PEER_DB_HASH_GET_RSP`

This message returns the result of APP calling `ble_conn_peer_db_hash_get` to get the information of the attribute database hash in the peer GATT database. The message data type is `ble_gap_peer_db_hash_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the attribute handle, database hash etc.

■ `BLE_CONN_EVT_PING_TO_VAL_GET_RSP`

This message returns the result of APP calling `ble_conn_ping_to_get` to get the BLE link ping timeout value. The message data type is `ble_gap_ping_tout_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the ping timeout value.

■ `BLE_CONN_EVT_PING_TO_INFO`

This message is used to actively notify APP after ping timeout. The message data type is `ble_gap_ping_tout_info_t`, including the connection index where the ping timeout occurs.

■ `BLE_CONN_EVT_PING_TO_SET_RSP`

This message returns the result of APP calling `ble_conn_ping_to_set` to set the ping timeout value. The message data type is `ble_gap_ping_tout_set_rsp_t`, including the set status, etc.

■ `BLE_CONN_EVT_RSSI_GET_RSP`

This message returns the result of APP calling `ble_conn_rssi_get` to get the RSSI of the last packet successfully received through the corresponding connection. The message data type is `ble_gap_peer_feats_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the RSSI, etc.

■ `BLE_CONN_EVT_CHANN_MAP_GET_RSP`

This message returns the result of APP calling `ble_conn_chann_map_get` to get the channel map used by the corresponding connection. The message data type is `ble_gap_chann_map_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the channel map array information.

■ `BLE_CONN_EVT_NAME_GET_IND`

This message is used to notify APP when the peer device tries to get the local name. The message data type is `ble_gap_name_get_ind_t`, including the start offset and the maximum name length of the name to return. APP can call `ble_conn_name_get_cfm` to reply.

■ `BLE_CONN_EVT_APPEARANCE_GET_IND`

This message is used to notify APP when the peer device tries to get the local appearance.

The message data type is `ble_gap_appearance_get_ind_t`. APP can call `ble_conn_appearance_get_cfm` to reply.

■ **BLE_CONN_EVT_SLAVE_PREFER_PARAM_GET_IND**

This message is used to notify APP when the peer device tries to get the local slave preferred parameter attribute. The message data type is `ble_gap_slave_prefer_param_get_ind_t`. APP can call `ble_conn_slave_prefer_param_get_cfm` to reply.

■ **BLE_CONN_EVT_NAME_SET_IND**

This message is used to notify APP when the peer device tries to set the local name. The message data type is `ble_gap_name_set_ind_t`, including the name length and name content to be set. APP can call `ble_conn_name_set_cfm` to reply.

■ **BLE_CONN_EVT_APPEARANCE_SET_IND**

This message is used to notify APP when the peer device tries to set the local appearance. The message data type is `ble_gap_appearance_set_ind_t`, including the appearance value to be set. APP can call `ble_conn_appearance_set_cfm` to reply.

■ **BLE_CONN_EVT_PARAM_UPDATE_IND**

This message is used to notify APP when the peer initiates the connection parameter update. The message data type is `ble_gap_conn_param_update_ind_t`, including parameters such as connection interval, latency, and supervision timeout that the peer wants to update. APP can call `ble_conn_param_update_cfm` to reply.

■ **BLE_CONN_EVT_PARAM_UPDATE_RSP**

This message returns the result of APP calling `ble_conn_param_update_req` to initiate the connection parameter update. The message type is `ble_gap_conn_param_update_rsp_t`, including the update status.

■ **BLE_CONN_EVT_PARAM_UPDATE_INFO**

This message is used to notify APP after the connection parameter update initiated by the peer or local device is completed. The message data type is `ble_gap_conn_param_info_t`, including the connection interval, latency, supervision timeout, etc. used after the update.

■ **BLE_CONN_EVT_PKT_SIZE_SET_RSP**

This message returns the result of APP calling `ble_conn_pkt_size_set` to set the size of packets sent by the local device. The message data type is `ble_gap_pkt_size_set_rsp_t`, including the set status.

■ **BLE_CONN_EVT_PKT_SIZE_INFO**

This message is used to notify APP after the packet size update initiated by the peer or local device is completed. The message data type is `ble_gap_pkt_size_info_t`, including the max tx octets, max tx time, max rx octets, and max rx time.

■ BLE_CONN_EVT_PHY_GET_RSP

This message returns the result of APP calling `ble_conn_phy_get` to get the PHY information used by the connection. The message data type is `ble_gap_phy_get_rsp_t`, including the obtained status.

■ BLE_CONN_EVT_PHY_SET_RSP

This message returns the result of APP calling `ble_conn_phy_set` to set the PHY used by the connection. The message data type is `ble_gap_phy_set_rsp_t`, including the set status.

■ BLE_CONN_EVT_PHY_INFO

This message is used to notify APP of the currently used PHY information after APP gets the connection PHY information and APP or the peer completes the setting of connection PHY. The message data type is `ble_gap_phy_info_t`, including the tx PHY and rx PHY information of the current connection.

■ BLE_CONN_EVT_LOC_TX_PWR_GET_RSP

This message returns the result of APP calling `ble_conn_local_tx_pwr_get` to get the local transmit power. The message data type is `ble_gap_local_tx_pwr_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the obtained PHY, the currently used transmit power on the corresponding PHY, and the maximum transmit power.

■ BLE_CONN_EVT_PEER_TX_PWR_GET_RSP

This message returns the result of APP calling `ble_conn_peer_tx_pwr_get` to get the peer transmit power. The message data type is `ble_gap_peer_tx_pwr_get_rsp_t`, including the obtained status. If the status is `BLE_ERR_NO_ERROR`, it also contains the obtained PHY, the transmit power on the corresponding PHY used by the peer, and power flags.

■ BLE_CONN_EVT_TX_PWR_RPT_CTRL_RSP

This message returns the result of APP calling `ble_conn_tx_pwr_report_ctrl` to set the transmit power report. The message data type is `ble_gap_tx_pwr_report_ctrl_rsp_t`, including the set status.

■ BLE_CONN_EVT_LOC_TX_PWR_RPT_INFO

This message is used to notify APP after APP calls `ble_conn_tx_pwr_report_ctrl` to enable the report when the local transmit power changes. The message data type is `ble_gap_tx_pwr_report_info_t`, including the PHY reported by the local device, the transmit power on the corresponding PHY, power flags, and changed transmit power delta.

■ BLE_CONN_EVT_PEER_TX_PWR_RPT_INFO

This message is used to notify APP after APP calls `ble_conn_tx_pwr_report_ctrl` to enable the report when the peer transmit power changes. The message data type is `ble_gap_tx_pwr_report_info_t`, including the PHY reported by the peer device, the transmit

power on the corresponding PHY, power flags, and changed transmit power delta.

- BLE_CONN_EVT_PATH_LOSS_CTRL_RSP

This message returns the result of APP calling `ble_conn_path_loss_ctrl` to set the path loss. The message data type is `ble_gap_path_loss_ctrl_rsp_t`, including the set status.

- BLE_CONN_EVT_PATH_LOSS_THRESHOLD_INFO

This message is used to notify APP after APP calls `ble_conn_path_loss_ctrl` to set the path loss and the path loss zone changes. The message data type is `ble_gap_path_loss_threshold_info_t`, including the current path loss value and the corresponding zone information.

- BLE_CONN_EVT_PER_SYNC_TRANS_RSP

This message returns the result of APP calling `ble_conn_per_adv_sync_trans` to sync transfer periodic advertising to the peer device. The message type is `ble_gap_per_adv_sync_trans_rsp_t`, including the transfer success or failure status.

2.5.2. `ble_conn_init`

Prototype: `ble_status_t ble_conn_init(void)`

Function: Initialize the BLE connection module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.5.3. `ble_conn_callback_register`

Prototype: `ble_status_t ble_conn_callback_register(ble_conn_evt_handler_t callback)`

Function: Register the callback function for processing BLE connection messages

Input parameter: `callback`, a function that processes BLE connection messages. For

the description of connection messages, see [Connection message type](#).

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.5.4. `ble_conn_callback_unregister`

Prototype: `ble_status_t ble_conn_callback_unregister(ble_conn_evt_handler_t callback)`

Function: Unregister the callback function from the BLE connection module

Input parameter: callback, a function that processes BLE connection messages

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.5.5. **ble_conn_connect**

Prototype: `ble_status_t ble_conn_connect(ble_gap_init_param_t *p_param,
ble_gap_local_addr_type_t own_addr_type,
ble_gap_addr_t *p_peer_addr_info, bool use_wl)`

Function: Initiate BLE connection

Input parameters: `p_param`, the parameter structure pointer used when initiating connections, including the connection interval, window, etc.

`own_addr_type`, the local address type used when creating connections

`p_peer_addr_info`, the peer device address information pointer

`use_wl`, indicating whether FAL is used; if yes, it will connect to the device in FAL instead of the address specified by `p_peer_addr_info`.

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After successful connection,

a `BLE_CONN_EVT_STATE_CHG` message is sent to the callback function,

and the state is `BLE_CONN_STATE_CONNECTED`. The connection index

included in the connection info can be used for subsequent operations.

2.5.6. **ble_conn_disconnect**

Prototype: `ble_status_t ble_conn_disconnect(uint8_t conidx, uint16_t reason)`

Function: Disconnect BLE connection

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection

`reason`, the reason for disconnection; use `BLE_ERROR_HL_TO_HCI`

(`BLE_LL_ERR_xxx`), and `BLE_LL_ERR_xxx` is the error code of the

LL group in `ble_err_t`

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After successful disconnection, a `BLE_CONN_EVT_STATE_CHG` message is sent to the callback function, and the state is `BLE_CONN_STATE_DISCONNECTED`

2.5.7. `ble_conn_connect_cancel`

Prototype: `ble_status_t ble_conn_connect_cancel(void)`

Function: Cancel the BLE connection being initiated

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.5.8. `ble_conn_sec_info_set`

Prototype: `ble_status_t ble_conn_sec_info_set(uint8_t conidx, uint8_t *p_local_csrk, uint8_t *p_peer_csrk, uint8_t pairing_lvl, uint8_t enc_key_present)`

Function: If APP manages security keys, after receiving the `BLE_CONN_EVT_STATE_CHG` message showing the connection state is `BLE_CONN_STATE_CONNECTED`, it should call this API to transfer key information to BLE stack.

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

`p_local_csrk`, local CSRK

`p_peer_csrk`, peer CSRK

`pairing_lvl`, pairing level

`enc_key_present`, which indicates whether encryption key is present

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.5.9. **ble_conn_peer_name_get**

Prototype: `ble_status_t ble_conn_peer_name_get(uint8_t conidx)`

Function: Get the name of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_NAME_GET_RSP` message is sent to the callback function

2.5.10. **ble_conn_peer_feats_get**

Prototype: `ble_status_t ble_conn_peer_feats_get(uint8_t conidx)`

Function: Get the features supported by the peer device that has established a connection

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_FEATS_GET_RSP` message is sent to the callback function

2.5.11. **ble_conn_peer_appearance_get**

Prototype: `ble_status_t ble_conn_peer_appearance_get(uint8_t conidx)`

Function: Get the appearance of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_APPEARANCE_GET_RSP` message is sent to the callback function

2.5.12. **ble_conn_peer_version_get**

Prototype: `ble_status_t ble_conn_peer_version_get(uint8_t conidx)`

Function: Get the version information of the peer device that has established a connection

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_VERSION_GET_RSP` message is sent to the callback function

2.5.13. **ble_conn_peer_slave_prefer_param_get**

Prototype: `ble_status_t ble_conn_peer_slave_prefer_param_get(uint8_t conidx)`

Function: Get the slave prefer parameters attribute of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_SLV_PRF_PARAM_GET_RSP` message is sent to the callback function

2.5.14. ble_conn_peer_addr_resolution_support_get

Prototype: `ble_status_t ble_conn_peer_addr_resolution_support_get(uint8_t conidx)`

Function: Get the address resolution support attribute of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_ADDR_RESLV_GET_RSP` message is sent to the callback function

2.5.15. ble_conn_peer_rpa_only_get

Prototype: `ble_status_t ble_conn_peer_rpa_only_get(uint8_t conidx)`

Function: Get the RPA only attribute of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure. After execution, a `BLE_CONN_EVT_PEER_RPA_ONLY_GET_RSP` message is sent to the callback function

2.5.16. ble_conn_peer_db_hash_get

Prototype: `ble_status_t ble_conn_peer_db_hash_get(uint8_t conidx)`

Function: Get the database hash attribute of the peer device that has established a connection in the GATT database

Input parameter: `conidx`, BLE connection index, which can be obtained in

the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_PEER_DB_HASH_GET_RSP`

message is sent to the callback function

2.5.17. **ble_conn_phy_get**

Prototype: `ble_status_t ble_conn_phy_get(uint8_t conidx)`

Function: Get the PHY being used by the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in

the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_PHY_GET_RSP`

message is sent to the callback function. If the PHY is successfully obtained,

a `BLE_CONN_EVT_PHY_INFO` message is also sent to the callback function

2.5.18. **ble_conn_phy_set**

Prototype: `ble_status_t ble_conn_phy_set(uint8_t conidx, uint8_t tx_phy, uint8_t rx_phy,
uint8_t phy_opt)`

Function: Set the PHY used by the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in

the connection success message

`tx_phy`, the PHY bitfield used by tx, which is composed of `ble_gap_le_phy_bf_t`

`rx_phy`, the PHY bitfield used by rx, which is composed of `ble_gap_le_phy_bf_t`

`phy_opt`, in the case of coded PHY, set the preference of S=2 or S=8

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_PHY_SET_RSP`
message is sent to the callback function. After the PHY is successfully set,
a `BLE_CONN_EVT_PHY_INFO` message is also sent to the callback function

2.5.19. `ble_conn_pkt_size_set`

Prototype: `ble_status_t ble_conn_pkt_size_set(uint8_t conidx, uint16_t tx_octets,
uint16_t tx_time)`

Function: Set the maximum packet size that an established connection can use
when transmitting

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

`tx_octets`, the maximum number of octets in the tx packet

`tx_time`, the maximum time for sending tx packets

Output parameter: None

Return value: Return 0 on successful execution, and return the error code
defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_PKT_SIZE_SET_RSP`
message is sent to the callback function. After packet size is successfully set,
a `BLE_CONN_EVT_PKT_SIZE_INFO` message is sent to the callback function.

2.5.20. `ble_conn_chann_map_get`

Prototype: `ble_status_t ble_conn_chann_map_get(uint8_t conidx)`

Function: Get the channel map used by the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code
defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_CHANN_MAP_GET_RSP`

message is sent to the callback function

2.5.21. **ble_conn_ping_to_get**

Prototype: `ble_status_t ble_conn_ping_to_get(uint8_t conidx)`

Function: Get the ping timeout value of the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code
defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_PING_TO_VAL_GET_RSP`
message is sent to the callback function

2.5.22. **ble_conn_ping_to_set**

Prototype: `ble_status_t ble_conn_ping_to_set(uint8_t conidx, uint16_t tout)`

Function: Set the ping timeout value of the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

`tout`, ping timeout value, in 10 ms

Output parameter: None

Return value: Return 0 on successful execution, and return the error code
defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_PING_TO_SET_RSP`
message is sent to the callback function

2.5.23. **ble_conn_rssi_get**

Prototype: `ble_status_t ble_conn_rssi_get(uint8_t conidx)`

Function: Get the rssi of the packet recently received on the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_RSSI_GET_RSP`

message is sent to the callback function.

2.5.24. `ble_conn_param_update_req`

Prototype: `ble_status_t ble_conn_param_update_req (uint8_t conidx, uint16_t interval,`

`uint16_t latency, uint16_t supv_to, uint16_t ce_len)`

Function: Set connection parameters of the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in

the connection success message

`interval`, the connection event period to be set, in 1.25 ms

`latency`, the maximum number of connection events for the master packet

that the slave does not need to listen to

`supv_to`, disconnection timeout, in 10 ms

`ce_len`, the length of connection events, in 0.625 ms

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_PARAM_UPDATE_RSP` message is sent to

the callback function. After the connection parameters are successfully updated,

a `BLE_CONN_EVT_PARAM_UPDATE_INFO`

message is also sent to the callback function

2.5.25. `ble_conn_per_adv_sync_trans`

Prototype: `ble_status_t ble_conn_per_adv_sync_trans (uint8_t conidx, uint8_t trans_idx,`

`uint16_t srv_data)`

Function: Forward periodic advertising information to the peer device that has

established a connection, so that it can directly initiate sync

Input parameter: conidx, BLE connection index, which can be obtained in

the connection success message

trans_idx, the index to be forwarded, which can be the index of periodic advertising created by the local device or the sync index after the local sync is successful

srv_data, the service data that APP can set

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure. After execution,

a BLE_CONN_EVT_PER_SYNC_TRANS_RSP

message is sent to the callback function

2.5.26. ble_conn_name_get_cfm

Prototype: ble_status_t ble_conn_name_get_cfm(uint8_t conidx, uint16_t status, uint16_t token, uint16_t cmpl_len, uint8_t *p_name, uint16_t name_len)

Function: This function is used to reply the request initiated by the peer device to get the local name after receiving the BLE_CONN_EVT_NAME_GET_IND message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in

the connection success message

status, the confirm status; if there is an error or exception,

fill in the error code; otherwise, fill in 0

token, message token, which is obtained in the

BLE_CONN_EVT_NAME_GET_IND message

cmpl_len, the total length of the local name

p_name, a pointer to the complete or partial content of the replied name

name_len, the length of the name in this reply. If the complete

name is replied, the length is equal to cmpl_len

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.27. ble_conn_appearance_get_cfm

Prototype: ble_status_t ble_conn_appearance_get_cfm(uint8_t conidx, uint16_t status,
uint16_t token, uint16_t appearance)

Function: This function is used to reply the request initiated by the peer device to get the local appearance after receiving the BLE_CONN_EVT_APPEARANCE_GET_IND message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in the connection success message

status, the confirm status; if there is an error or exception,

fill in the error code; otherwise, fill in 0

token, which is obtained in the

BLE_CONN_EVT_APPEARANCE_GET_IND message

appearance, the local appearance replied

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.28. ble_conn_slave_prefer_param_get_cfm

Prototype: ble_status_t ble_conn_slave_prefer_param_get_cfm (uint8_t conidx,
uint16_t status, uint16_t token, ble_gap_prefer_periph_param_t *p_param)

Function: This function is used to reply the request initiated by the peer device to get the slave prefer parameter after receiving the BLE_CONN_EVT_SLAVE_PREFER_PARAM_GET_IND message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in the connection success message

status, the confirm status; if there is an error or exception,

fill in the error code; otherwise, fill in 0

token, which is obtained in the

BLE_CONN_EVT_SLAVE_PREFER_PARAM_GET_IND message

p_param, slave prefer parameter structure pointer, including the
interval, latency, etc.

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.29. ble_conn_name_set_cfm

Prototype: ble_status_t ble_conn_name_set_cfm (uint8_t conidx, uint16_t status,
uint16_t token)

Function: This function is used to reply the request initiated by the peer device to set
the local name after receiving the BLE_CONN_EVT_NAME_SET_IND
message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in
the connection success message
status, the confirm status; if there is an error or exception,
fill in the error code; otherwise, fill in 0
token, which is obtained in
the BLE_CONN_EVT_NAME_SET_IND message

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.30. ble_conn_appearance_set_cfm

Prototype: ble_status_t ble_conn_appearance_set_cfm(uint8_t conidx, uint16_t status,
uint16_t token)

Function: This function is used to reply the request initiated by the peer device to set the local
appearance after receiving the
BLE_CONN_EVT_APPEARANCE_SET_IND message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in
the connection success message
status, the confirm status; if there is an error or exception,

fill in the error code; otherwise, fill in 0

token, which is obtained in

the BLE_CONN_EVT_APPEARANCE_SET_IND message

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.31. ble_conn_param_update_cfm

Prototype: ble_status_t ble_conn_param_update_cfm(uint8_t conidx, bool accept,
uint16_t ce_len_min, uint16_t ce_len_max)

Function: This function is used to reply the connection parameter update request

initiated by the peer device after receiving

the BLE_CONN_EVT_PARAM_UPDATE_IND message in the callback function

Input parameter: conidx, BLE connection index, which can be obtained in

the connection success message

accept, true means to accept the connection parameter update request;

otherwise, return false

ce_len_min, the minimum time of connection events, in 0.625 ms

ce_len_max, the maximum time of connection events, in 0.625 ms

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.5.32. ble_conn_local_tx_pwr_get

Prototype: ble_status_t ble_conn_local_tx_pwr_get(uint8_t conidx,
ble_gap_phy_pwr_value_t phy)

Function: Get the local transmit power on the corresponding PHY of

the established connection

Input parameter: conidx, BLE connection index, which can be obtained in

the connection success message

phy, the PHY corresponding to the obtained power

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_LOC_TX_PWR_GET_RSP`

message is sent to the callback function

2.5.33. `ble_conn_peer_tx_pwr_get`

Prototype: `ble_status_t ble_conn_peer_tx_pwr_get (uint8_t conidx,`

`ble_gap_phy_pwr_value_t phy)`

Function: Get the peer transmit power used on the corresponding PHY of
the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

`phy`, the PHY corresponding to the obtained power

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_CONN_EVT_PEER_TX_PWR_GET_RSP`

message is sent to the callback function

2.5.34. `ble_conn_tx_pwr_report_ctrl`

Prototype: `ble_status_t ble_conn_tx_pwr_report_ctrl(uint8_t conidx, uint8_t local_en,`

`uint8_t remote_en)`

Function: Set whether to send a notification to APP when the local or peer
transmit power changes on the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in
the connection success message

`local_en`, whether to notify APP when the local transmit power changes

`remote_en`, whether to notify APP when the peer transmit power changes

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_TX_PWR_RPT_CTRL_RSP`
message is sent to the callback function. If local enable is successfully set,
when the local transmit power changes,
a `BLE_CONN_EVT_LOC_TX_PWR_RPT_INFO`
message is sent to the callback function. If remote enable is successfully set,
when the peer tx power changes,
a `BLE_CONN_EVT_PEER_TX_PWR_RPT_INFO`
message is sent to the callback function

2.5.35. `ble_conn_path_loss_ctrl`

Prototype: `ble_status_t ble_conn_path_loss_ctrl (uint8_t conidx, uint8_t enable,
uint8_t high_threshold, uint8_t high_hysteresis, int8_t low_threshold,
uint8_t low_hysteresis, uint16_t min_time)`

Function: Set the path loss notification on the established connection

Input parameter: `conidx`, BLE connection index, which can be obtained in

the connection success message

`enable`, whether to notify of path

`loss high_threshold`, the threshold of path loss in the high zone

`high_hysteresis`, the hysteresis value of the high threshold

`low_threshold`, the threshold of path loss in the low zone

`low_hysteresis`, the hysteresis value of the low threshold

`min_time`, the minimum number of connection events to stay after

the path changes

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,
a `BLE_CONN_EVT_PATH_LOSS_CTRL_RSP`
message is sent to the callback function. If it is successfully set to enable,
when the path zone changes,

a `BLE_CONN_EVT_PATH_LOSS_THRESHOLD_INFO` message is sent to the callback function

2.5.36. `ble_conn_enable_central_feat`

Prototype: `ble_status_t ble_conn_enable_central_feat(uint8_t conidx)`

Function: When the device serves as peripheral, it actively obtains and configures the central gap service information. When `cfg.att` is set to `BLE_GAP_ATT_CLI_DIS_AUTO_FEAT_EN` bit in `ble_adp_cfg`, this api needs to be invoked after the connection is successful. Otherwise, this api is not used.

Input parameter: `conidx`, BLE connection index, which can be obtained in the connection success message

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.6. BLE security API

The header file is `ble_sec.h`.

The BLE security module mainly provides interfaces required for interaction during pairing, authentication, encryption, and other processes.

2.6.1. Security message type

APP can register a callback function with the BLE security module, and the BLE security module will send the following event messages to APP through the callback function.

■ `BLE_SEC_EVT_PAIRING_REQ_IND`

This message is used to notify APP after the pairing request initiated by the peer device is received. The message data type is `ble_gap_pairing_req_ind_t`, including the peer authentication request level, etc. APP can call `ble_sec_pairing_req_cfm` to reply.

■ `BLE_SEC_EVT_LTK_REQ_IND`

This message is used to get the long term key of the paired device from APP during authentication. The message data type is `ble_gap_ltk_req_ind_t`, including the LTK size information. APP can call `ble_sec_ltk_req_cfm` to reply.

■ `BLE_SEC_EVT_KEY_DISPLAY_REQ_IND`

This message is used to get the PIN CODE from APP during pairing. The message data type is `ble_gap_tk_req_ind_t`, including the connection index information. APP can call `ble_sec_key_display_enter_cfm` to reply.

■ `BLE_SEC_EVT_KEY_ENTER_REQ_IND`

This message is used to notify APP when the user is required to enter the passkey during pairing. The message data type is `ble_gap_tk_req_ind_t`, including the connection index information. APP can call `ble_sec_key_display_enter_cfm` to reply.

■ `BLE_SEC_EVT_KEY_OOB_REQ_IND`

This message is used to notify APP when APP is required to use OOB data as the temp key. The message data type is `ble_gap_tk_req_ind_t`, including the connection index information. APP can call `ble_sec_oob_req_cfm` to reply.

■ `BLE_SEC_EVT_NUMERIC_COMPARISON_IND`

This message is used to notify APP when the user is required to compare the generated number on both sides during pairing. The message data type is `ble_gap_nc_ind_t`, including the number to be compared. APP can call `ble_sec_nc_cfm` to reply.

■ `BLE_SEC_EVT_IRK_REQ_IND`

This message is used to notify APP when the local IRK needs to be obtained and distributed during pairing. The message data type is `ble_gap_irk_req_ind_t`, including the connection index information. APP can call the `ble_sec_irk_req_cfm` function to reply.

■ `BLE_SEC_EVT_CSRK_REQ_IND`

This message is used to notify APP when the local CSRK needs to be obtained and distributed during pairing. The message data type is `ble_gap_csrk_req_ind_t`, including the connection index information. APP can call the `ble_sec_csrk_req_cfm` function to reply.

■ `BLE_SEC_EVT_OOB_DATA_REQ_IND`

This message is used to get OOB data from APP when using the OOB mode during pairing. The message data type is `ble_gap_oob_data_req_ind_t`, including the connection index information. APP can call the `ble_sec_oob_data_req_cfm` function to reply.

■ `BLE_SEC_EVT_PAIRING_SUCCESS_INFO`

This message is used to notify APP after the pairing is successful. The message data type is `ble_sec_pairing_success_t`, including whether it is a secure connection, the pairing level, etc.

■ `BLE_SEC_EVT_PAIRING_FAIL_INFO`

This message is used to notify APP when the pairing fails. The message data type is `ble_sec_pairing_fail_t`, including the reason for pairing failure, etc.

■ `BLE_SEC_EVT_SECURITY_REQ_INFO`

This message is used to notify APP when the master receives the security request initiated

by the peer slave. The message data type is `ble_sec_security_req_info_t`, including the authentication request level and other information of the peer device. APP can decide to initiate encryption or pairing based on whether there is a LTK from the peer device after receiving the message.

■ BLE_SEC_EVT_ENCRYPT_REQ_IND

This message is used to notify APP after the encryption request initiated by the peer device is received. The message data type is `ble_gap_encrypt_req_ind_t`, including the ediv, random number, etc. APP can call `ble_sec_encrypt_req_cfm` to reply.

■ BLE_SEC_EVT_ENCRYPT_INFO

This message is used to notify APP after encryption is completed. The message data type is `ble_sec_encrypt_info_t`, including the encryption success or failure status. If successful, it also contains the pairing level and other information.

■ BLE_SEC_EVT_OOB_DATA_GEN_INFO

This message is used to notify APP after APP calls `ble_sec_oob_data_gen` to generate a set of OOB data. The message data type is `ble_sec_oob_data_info_t`, including the generated OOB data.

■ BLE_SEC_EVT_KEY_PRESS_NOTIFY_RSP

This message returns the result of APP calling `ble_sec_key_press_notify`. The message data type is `ble_gap_key_press_ntf_rsp_t`, including the status of sending key press notification.

■ BLE_SEC_EVT_KEY_PRESS_INFO

This message is used to notify APP after the key press notification of the peer device is received. The message data type is `ble_gap_key_pressed_info_t`, including the key press type and other information of the peer device.

2.6.2. **ble_sec_init**

Prototype: `ble_status_t ble_sec_init(void)`

Function: Initialize the BLE security module.

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.6.3. **ble_sec_callback_register**

Prototype: `ble_status_t ble_sec_callback_register(ble_sec_evt_handler_t callback)`

Function: The interface is used to register the event message handler with

the BLE security module.

Input parameter: callback, callback handler. For the description of security messages,

See [Security message type](#).

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure.

2.6.4. ble_sec_callback_unregister

Prototype: ble_status_t ble_sec_callback_unregister(ble_sec_evt_handler_t callback)

Function: Unregister the callback function from BLE security module

Input parameter: callback, callback function to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure.

2.6.5. ble_sec_security_req

Prototype: ble_status_t ble_sec_security_req(uint8_t conidx, uint8_t auth)

Function: Send a security request message for active pairing as a slave.

Input parameter: conidx, connection index

auth, indicating the pairing security type. Refer to the enum

ble_gap_auth_mask_t.

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure.

2.6.6. ble_sec_bond_req

Prototype: ble_status_t ble_sec_bond_req(uint8_t conidx,

ble_gap_pairing_param_t *p_param, uint8_t sec_req_level)

Function: Send a pairing request message for active pairing as a master, or respond to

the security request from the peer slave to initiate pairing after receiving

the BLE_SEC_EVT_SECURITY_REQ_INFO message

Input parameter: conidx, connection index

p_param, the parameter of the pairing request message. Refer to

the structure ble_gap_pairing_param_t

sec_req_level, security request level. Refer to the enum ble_gap_sec_req_t

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure.

2.6.7. ble_sec_encrypt_req

Prototype: ble_status_t ble_sec_encrypt_req(uint8_t conidx, ble_gap_ltk_t *p_peer_ltk)

Function: Send an encryption request when there is a LTK from the peer device

Input parameter: conidx, connection index

p_peer_ltk, the peer ltk

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.8. ble_sec_key_press_notify

Prototype: ble_status_t ble_sec_key_press_notify(uint8_t conidx, uint8_t type)

Function: Send a keypress notify message

Input parameter: conidx, connection index

type, 0: Passkey entry started
1: Passkey digit entered
2: Passkey digit erased
3: Passkey cleared
4: Passkey entry completed

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure. After execution,

a BLE_SEC_EVT_KEY_PRESS_NOTIFY_RSP

message is sent to the callback function

2.6.9. ble_sec_key_display_enter_cfm

Prototype: ble_status_t ble_sec_key_display_enter_cfm(uint8_t conidx, bool accept,
uint32_t passkey)

Function: This function is used to reply PIN CODE or passkey during pairing after receiving BLE_SEC_EVT_KEY_DISPLAY_REQ_IND or BLE_SEC_EVT_KEY_ENTER_REQ_IND in the callback function.

Input parameter: conidx, connection index
accept, whether to accept the request
passkey, the value range is 000000-999999

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.10. ble_sec_oob_req_cfm

Prototype: ble_status_t ble_sec_oob_req_cfm(uint8_t conidx, bool accept, uint8_t *p_key)

Function: This function is used to reply OOB TK during pairing after receiving the BLE_SEC_EVT_KEY_OOB_REQ_IND message in the callback function

Input parameter: conidx, connection index
accept, whether to accept the request
p_key, 128-bit key value

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.11. ble_sec_nc_cfm

Prototype: ble_status_t ble_sec_nc_cfm(uint8_t conidx, bool accept)

Function: This function is used to reply the results of numeric comparison during pairing after receiving the BLE_SEC_EVT_NUMERIC_COMPARISON_IND message in the callback function

Input parameter: conidx, connection index

accept, whether the results of numeric comparison are consistent

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure.

2.6.12. ble_sec_ltk_req_cfm

Prototype: ble_status_t ble_sec_ltk_req_cfm(uint8_t conidx, uint8_t accept,
ble_gap_ltk_t *p_ltk)

Function: This function is used to reply the local LTK information or reject the request
after receiving the BLE_SEC_EVT_LTK_REQ_IND message in the callback function

Input parameter: conidx, connection index

accept, whether to accept the request

p_ltk, a pointer to the ltk value

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure.

2.6.13. ble_sec_irk_req_cfm

Prototype: ble_status_t ble_sec_irk_req_cfm(uint8_t conidx, uint8_t accept,
ble_gap_irk_t *p_irk)

Function: The function is used to reply the local IRK information or reject the request
after receiving the BLE_SEC_EVT_IRK_REQ_IND message in the callback function

Input parameter: conidx, connection index

accept, whether to accept the request

p_irk, a pointer to the irk value

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure.

2.6.14. ble_sec_csrk_req_cfm

Prototype: ble_status_t ble_sec_csrk_req_cfm(uint8_t conidx, uint8_t accept,
ble_gap_csrk_t *p_csrk)

Function: This function is used to reply the local CSRK information or reject the request after receiving the BLE_SEC_EVT_CSRK_REQ_IND message in the callback function

Input parameter: conidx, connection index
accept, whether to accept the request
p_csrk, a pointer to the csrk value

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.15. ble_sec_encrypt_req_cfm

Prototype: ble_status_t ble_sec_encrypt_req_cfm(uint8_t conidx, bool found, uint8_t *p_ltk,
uint8_t key_size)

Function: This function is used to reply the local LTK information or reject the request during encryption after receiving the BLE_SEC_EVT_ENCRYPT_REQ_IND message in the callback function

Input parameter: conidx, connection index
found, whether the key exists
p_ltk, a pointer to the local ltk value
key_size, key size

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.16. ble_sec_pairing_req_cfm

Prototype: ble_status_t ble_sec_pairing_req_cfm(uint8_t conidx, uint8_t accept,
ble_gap_pairing_param_t *p_param, uint8_t sec_req_lvl)

Function: This function is used to reply the pairing response to the peer device for setting or reject the request after receiving the BLE_SEC_EVT_PAIRING_REQ_IND message in the callback function

Input parameter: conidx, connection index

accept, whether to accept the request

p_param, the parameter of the pairing response message.

Refer to the structure ble_gap_pairing_param_t

sec_req_level, security request level. Refer to the enum ble_gap_sec_req_t

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.17. ble_sec_oob_data_req_cfm

Prototype: ble_status_t ble_sec_oob_data_req_cfm(uint8_t conidx, uint8_t accept,
uint8_t *p_conf, uint8_t *p_rand)

Function: This function is used to reply the local OOB information or reject the request during pairing after receiving the BLE_SEC_EVT_OOB_DATA_REQ_IND message in the callback function.

Input parameter: conidx, connection index

accept, whether to accept the request

p_conf, the peer confirm value

p_rand, the peer random value

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in ble_status_t on failure.

2.6.18. ble_sec_oob_data_gen

Prototype: ble_status_t ble_sec_oob_data_gen(void)

Function: This function is used to generate a set of OOB data.

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After OOB data is successfully generated, a `BLE_SEC_EVT_OOB_DATA_GEN_INFO` message is sent to the callback function

2.7. BLE list API

The header file is `ble_list.h`.

The BLE list module mainly provides interfaces for operating FAL, RAL, and PAL, including operations such as adding devices to the list, deleting devices from the list, and clearing the list.

2.7.1. List message type

■ BLE_LIST_EVT_OP_RSP

This message returns the result of APP calling the function `ble_fal_op`, `ble_fal_list_set`, `ble_fal_list_clear`, `ble_ral_op`, `ble_ral_list_set`, `ble_ral_list_clear`, `ble_pal_op`, `ble_pal_list_set`, or `ble_pal_list_clear` to operate the list. The message data type is `ble_list_data_t`, including the list type, op type, etc. Determine which list operation the reply is for according to the type in the data.

■ BLE_LIST_EVT_LOC_RPA_GET_RSP

This message returns the result of APP calling `ble_loc_rpa_get` to get the local resolvable address. The message data type is `ble_list_data_t`; the list type is `BLE_RAL_TYPE`, and the op type is `GET_LOC_RPA`.

■ BLE_LIST_EVT_PEER_RPA_GET_RSP

This message returns the result of APP calling `ble_peer_rpa_get` to get the peer resolvable address. The message data type is `ble_list_data_t`; the list type is `BLE_RAL_TYPE`, and the op type is `GET_PEER_RPA`.

2.7.2. ble_list_init

Prototype: `ble_status_t ble_list_init(void)`

Function: Initialize the BLE list module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.7.3. `ble_list_callback_register`

Prototype: `ble_status_t ble_list_callback_register(ble_list_evt_handler_t callback)`

Function: Register the callback function for processing BLE list messages

Input parameter: `callback`, a function that processes BLE list messages.

For the description of list messages, see [List message type](#).

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.7.4. `ble_list_callback_unregister`

Prototype: `ble_status_t ble_list_callback_unregister(ble_list_evt_handler_t callback)`

Function: Unregister the callback function from BLE list module

Input parameter: `callback`, callback function to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.7.5. `ble_fal_op`

Prototype: `ble_status_t ble_fal_op(ble_gap_addr_t *p_addr_info, bool add)`

Function: Add the specified device to or remove it from the filter accept list

Input parameter: `p_addr_info`, device address pointer

`add`, true means to add to FAL, false means to remove from FAL

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution, a `BLE_LIST_EVT_OP_RSP`

message is sent to the callback function; list type is `BLE_FAL_TYPE`,

and op type is `RMV_DEVICE_FROM_LIST` or `ADD_DEVICE_TO_LIST`

2.7.6. `ble_fal_list_set`

Prototype: `ble_status_t ble_fal_list_set(uint8_t num, ble_gap_addr_t *p_addr_info)`

Function: Set the filter accept list. This operation will update the whole FAL to

the specified content

Input parameter: num, the number of devices that need to be set to FAL

p_addr_info, device array, which contains the information of
num addresses

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure. After execution,
a BLE_LIST_EVT_OP_RSP message is sent to the callback function;
list type is BLE_FAL_TYPE, and op type is SET_DEVICES_TO_LIST

2.7.7. ble_fal_clear

Prototype: ble_status_t ble_fal_clear(void)

Function: Clear the filter accept list

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in ble_status_t on failure. After execution,
a BLE_LIST_EVT_OP_RSP message is sent to the callback function,
list type is BLE_FAL_TYPE, and op type is CLEAR_DEVICE_LIST

2.7.8. ble_fal_size_get

Prototype: uint8_t ble_fal_size_get(void)

Function: Get the maximum number of elements in the filter accept list

Input parameter: None

Output parameter: None

Return value: The maximum number of elements in the filter accept list

2.7.9. ble_ral_op

Prototype: ble_status_t ble_ral_op(ble_gap_ral_info_t *p_ral_info, bool add)

Function: Add the specified device to or remove it from the resolving list

Input parameter: `p_ral_info`, RAL structure pointer, including the identity address, IRK, etc.

`add`, true means to add to RAL, false means to remove from RAL

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function;

list type is `BLE_RAL_TYPE`, and op type is

`RMV_DEVICE_FROM_LIST` or `ADD_DEVICE_TO_LIST`

2.7.10. `ble_ral_list_set`

Prototype: `ble_status_t ble_ral_list_set(uint8_t num, ble_gap_ral_info_t *p_ral_info)`

Function: Set the resolving list. This operation will update the whole RAL to

the specified content

Input parameter: `num`, the number of devices that need to be set to RAL

`p_ral_info`, RAL structure array, which contains `num` RAL structures

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function;

list type is `BLE_RAL_TYPE`, and op type is `SET_DEVICES_TO_LIST`

2.7.11. `ble_ral_clear`

Prototype: `ble_status_t ble_ral_clear(void)`

Function: Clear the resolving list

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function;

list type is `BLE_RAL_TYPE`, and op type is `CLEAR_DEVICE_LIST`

2.7.12. ble_ral_size_get

Prototype: `uint8_t ble_ral_size_get(void)`

Function: Get the maximum number of elements in the resolving list

Input parameter: None

Output parameter: None

Return value: The maximum number of elements in the resolving list

2.7.13. ble_loc_rpa_get

Prototype: `ble_status_t ble_loc_rpa_get(uint8_t *p_peer_id, uint8_t peer_id_type)`

Function: Get the local resolvable private address currently used for the specified device

Input parameter: `p_peer_id`, the identity address of the specified device

`peer_id_type`, the identity address type of the specified device

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_LOC_RPA_GET_RSP` message is sent to

the callback function.

2.7.14. ble_peer_rpa_get

Prototype: `ble_status_t ble_peer_rpa_get(uint8_t *p_peer_id, uint8_t peer_id_type)`

Function: Get the resolvable private address currently used for the specified device

Input parameter: `p_peer_id`, the identity address of the specified device

`peer_id_type`, the identity address type of the specified device

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_PEER_RPA_GET_RSP` message is sent to

the callback function

2.7.15. **ble_pal_op**

Prototype: `ble_status_t ble_pal_op(ble_gap_pal_info_t *p_pal_info, bool add)`

Function: Add the specified device to or remove it from the periodic advertising list

Input parameter: `p_pal_info`, PAL structure pointer, including the address, SID, etc.

`add`, true means to add to PAL, false means to remove from PAL

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution, a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function; list type is `BLE_PAL_TYPE`, and op type is `RMV_DEVICE_FROM_LIST` or `ADD_DEVICE_TO_LIST`

2.7.16. **ble_pal_list_set**

Prototype: `ble_status_t ble_pal_list_set(uint8_t num, ble_gap_pal_info_t *p_pal_info)`

Function: Set the periodic advertising list. This operation will update the whole PAL to the specified content

Input parameter: `num`, the number of devices that need to be set to PAL

`p_pal_info`, PAL structure array, which contains `num` PAL structures

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined

in `ble_status_t` on failure. After execution, a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function; list type is `BLE_PAL_TYPE`, and op type is `SET_DEVICES_TO_LIST`

2.7.17. **ble_pal_clear**

Prototype: `ble_status_t ble_pal_clear(void)`

Function: Clear the periodic advertising list

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_LIST_EVT_OP_RSP` message is sent to the callback function;

list type is `BLE_PAL_TYPE`, and op type is `CLEAR_DEVICE_LIST`

2.7.18. `ble_pal_size_get`

Prototype: `uint8_t ble_pal_size_get(void)`

Function: Get the maximum number of elements in the periodic advertising list

Input parameter: None

Output parameter: None

Return value: The maximum number of elements in the periodic advertising list

2.8. BLE periodic sync API

The header file is `ble_per_sync.h`.

The BLE periodic sync module mainly provides interfaces for synchronizing periodic advertising, reporting received periodic advertising data, etc.

2.8.1. Periodic sync message type

APP can register a callback function with the BLE periodic sync module, and the BLE protocol stack will send the following event message to APP through the callback function.

■ `BLE_PER_SYNC_EVT_STATE_CHG`

This message is sent to the callback function when the periodic sync state changes. The message data type is `ble_per_sync_state_chg_t`, including the new state and the reason for change.

■ `BLE_PER_SYNC_EVT_REPORT`

This message is sent to the callback function after the periodic advertising report is received. The message data type is `ble_gap_adv_report_info_t`, including the device address for sending periodic advertising, the sent PHY, advertising data, etc.

■ `BLE_PER_SYNC_EVT_ESTABLISHED`

This message is sent to the callback function after the periodic advertising is synchronized. The message data type is `ble_per_sync_established_t`, including the the PHY of synchronized periodic advertising, interval, SID, etc.

■ `BLE_PER_SYNC_EVT_RPT_CTRL_RSP`

This message returns the result of APP calling `ble_per_sync_report_ctrl` to set the report content. The message data type is `ble_per_sync_rpt_ctrl_rsp_t`, including the set status.

2.8.2. `ble_per_sync_init`

Prototype: `ble_status_t ble_per_sync_init(void)`

Function: Initialize the BLE periodic sync module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.8.3. `ble_per_sync_callback_register`

Prototype: `ble_status_t ble_per_sync_callback_register(
ble_per_sync_evt_handler_t callback)`

Function: Register the callback function that processes periodic sync messages.

For the description of per sync messages, see [Periodic sync message type](#).

Input parameter: callback, callback function that processes periodic sync messages

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.8.4. `ble_per_sync_callback_unregister`

Prototype: `ble_status_t ble_per_sync_callback_unregister(
ble_per_sync_evt_handler_t callback)`

Function: Unregister the callback function from BLE periodic sync module

Input parameter: callback, callback function to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.8.5. `ble_per_sync_start`

Prototype: `ble_status_t ble_per_sync_start (ble_gap_local_addr_type_t own_addr_type,
ble_gap_per_sync_param_t *p_param)`

Function: Start periodic sync

Input parameter: `own_addr_type`, the local address type used in the sync process

`p_param`, periodic sync parameter structure pointer

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_PER_SYNC_EVT_STATE_CHG` message is sent to

the callback function. If successfully synchronized,

a `BLE_PER_SYNC_EVT_ESTABLISHED` message is also sent to

the callback function, and a `BLE_PER_SYNC_EVT_REPORT`

message is sent to report the received data

2.8.6. `ble_per_sync_cancel`

Prototype: `ble_status_t ble_per_sync_cancel (void)`

Function: Cancel the ongoing periodic sync process

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_PER_SYNC_EVT_STATE_CHG`

message is sent to the callback function

2.8.7. `ble_per_sync_terminate`

Prototype: `ble_status_t ble_per_sync_terminate (uint8_t sync_idx)`

Function: Abort the periodic sync train that has been successfully synchronized

Input parameter: `sync_idx`, sync index

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure. After execution,

a `BLE_PER_SYNC_EVT_STATE_CHG`

message is sent to the callback function

2.8.8. **ble_per_sync_ctrl**

Prototype: `ble_status_t ble_per_sync_ctrl(uint8_t sync_idx, uint8_t ctrl)`

Function: Modify the content of notification reported after successful synchronization

Input parameter: `sync_idx`, sync index

`ctrl`, periodic sync report control bit, which is composed of bits in

`ble_per_sync_rpt_ctrl_bit_t`

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

After the setting, a `BLE_PER_SYNC_EVT_RPT_CTRL_RSP`

message is sent to the callback function

2.9. **BLE storage API**

The header file is `ble_storage.h`. The module uses flash to store and manage the bond information of the peer device, including `peer_irk`, `peer_ltk`, `peer_csrk`, `local_irk`, `local_ltk`, `local_csrk`, etc.

The macro definition `BLE_PEER_NUM_MAX` in the header file is used to define the maximum number of peer devices. When the number of peer devices stored has reached the upper limit while new peer information needs to be stored, use the LRU algorithm to delete the oldest peer information that has not been used.

2.9.1. **ble_storage_init**

Prototype: `ble_status_t ble_storage_init(void)`

Function: Initialize the storage module. To get all the peer information from flash,

call the function once during initialization

Input parameter: None

Output parameter: None

Return value: Return 0 on successful execution, and return

the error code defined in `ble_status_t` on failure.

2.9.2. **ble_peer_data_bond_store**

Prototype: `ble_status_t ble_peer_data_bond_store(ble_gap_addr_t *addr,`

```
ble_gap_sec_bond_data_t *bond_data)
```

Function: The function is used to store the bond information of the peer device, which will also be saved in flash. If the bond information with the same index already exists, it will be updated and saved. If `keys_user_mgr` is false during BLE adapter config, the BLE security will automatically store the bond information, and APP does not need to perform related operations.

Input parameter: `addr`, the address of the connected device. If `bond_data` does not contain identity `addr`, the address will be stored as an index. If `bond_data` contains identity `addr`, identity `addr` will be stored as an index, and this address will not work; however, it cannot be empty

`bond_data`: the bond information needs to store

Output parameter: None

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.9.3. **ble_peer_data_bond_load**

```
Prototype: ble_status_t ble_peer_data_bond_load(ble_gap_addr_t *addr,  
ble_gap_sec_bond_data_t *bond_data)
```

Function: The function is used to get bond information

Input parameter: `addr`, which can be identity `addr` or RPA, with the address as an index to get information

Output parameter: `bond_data`, the obtained bond information

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.9.4. **ble_peer_data_delete**

```
Prototype: ble_status_t ble_peer_data_delete(ble_gap_addr_t *addr)
```

Function: The function is used to delete the peer information corresponding to the specified `addr`, and the content in flash will also be deleted.

Input parameter: `addr`, which can be identity `addr` or RPA, with the address as an index

to delete the peer information

Output parameter: None

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure.

2.9.5. **ble_peer_all_addr_get**

Prototype: `ble_status_t ble_peer_all_addr_get(uint8_t *num, ble_gap_addr_t *id_addrs)`

Function: The function is used to get the identity addr of all peer devices under

the storage module

Input parameter: `num`, the `num` pointer value indicates the maximum number of peer

devices that need to be obtained, which cannot

exceed `BLE_PEER_NUM_MAX` and determines the memory size of

the `id_addrs` pointer to be `num*sizeof(ble_gap_addr_t)`

Output parameter: `num`, whose value is the actual number obtained

`id_addrs`, the `id_addrs` pointer stores the actually obtained peer identity

addr

Return value: Return 0 on successful execution, and return the error code

defined in `ble_status_t` on failure.

2.9.6. **ble_svc_data_save**

Prototype: `ble_status_t ble_svc_data_save(uint8_t conn_idx, uint16_t data_id , uint32_t len,`

`uint8_t *p_data)`

Function: The function is provided to the upper-layer BLE service to store data related to the

service of the connected device to the flash.

Input parameter: `conn_idx`, BLE connection index, which can be obtained in

the connection success message

`data_id`, identifies service information and is defined by the upper-layer APP

`len`, length of `p_data`

`p_data`, store service information pointer

Output parameter: None

Return value: Return 0 on successful execution, and return the error code
defined in `ble_status_t` on failure.

2.9.7. **ble_svc_data_load**

Prototype: `ble_status_t ble_svc_data_load(uint8_t conn_idx, uint16_t data_id, void **pp_data, uint32_t *p_len)`

Function: The function is provided to the upper-layer BLE service to obtain data about the service stored in the flash on the connected device.

Input parameter: `conn_idx`, BLE connection index, which can be obtained in the connection success message

`data_id`, identifies service information and is defined by the upper-layer APP

Output parameter: `pp_data`, the obtained service information pointer

`p_len`, length of `pp_data`

Return value: Return 0 on successful execution, and return the error code defined in `ble_status_t` on failure.

2.10. **BLE gatts API**

The header file is `ble_gatts.h`.

The BLE GATT server module mainly provides interfaces for registering/deleting GATT service, sending notification/indication to the client, etc.

2.10.1. **gatts message type**

BLE services can register a callback function with the BLE GATT server module, and the BLE GATT server module will send the following event messages to BLE services through the callback function.

■ BLE_SRV_EVT_SVC_ADD_RSP

This message returns the result of calling the `ble_gatts_svc_add` function to add a service to the GATT server module. The message data type is `ble_gatts_svc_add_rsp_t`, including the status of the added service. If the status is 0, it also contains the assigned service ID and the start handle value of the service in the database.

■ BLE_SRV_EVT_SVC_RMV_RSP

This message returns the result of calling the `ble_gatts_svc_rmv` function to remove a service

from the GATT server module. The message data type is `ble_gatts_svc_rmv_rsp_t`, including the status of the removed service and the service ID.

■ BLE_SRV_EVT_CONN_STATE_CHANGE_IND

This message is sent to the callback function when the device connection state changes. The message data type is `ble_gatts_conn_state_change_ind_t`, including the connection status. If the connection state is connected, the connection index and address information of the peer device will be included; If the connection state is disconnected, the reason for disconnection will also be included.

■ BLE_SRV_EVT_GATT_OPERATION

This message is sent to the callback function when interacting with the peer GATT client. The message data type is `ble_gatts_op_info_t`, including the subevent, connection index of interacted connection, and message data of different subevents. The message includes the following subevents:

● BLE_SRV_EVT_READ_REQ

When the peer client initiates the attribute read request, this subevent will be notified to the callback function. The corresponding data type is `ble_gatts_read_req_t`, including the attribute index to be read, and the offset and the maximum length of the attribute value. At the same time, the message also contains `pending_cfm` flag, through which the upper layer can determine whether to directly reply to the peer client with the read result through the GATT server module after the message is process by the callback function. If required, copy the data to the pre-allocated location (the maximum length) of the server module; otherwise, set `pending_cfm` to true, and call `ble_gatts_svc_attr_read_cfm` to reply as required.

● BLE_SRV_EVT_WRITE_REQ

When the peer client initiates the attribute write request, this subevent will notify the callback function by using the data type of `ble_gatts_write_req_t`, including the attribute index to be written, and the offset, length, and content of the written data. At the same time, the message also contains `pending_cfm` flag, through which the upper layer can determine whether to directly reply with the write result through the GATT server module after the message is process by the callback function. If not required, set `pending_cfm` to true, and call `ble_gatts_svc_attr_write_cfm` to reply as required.

● BLE_SRV_EVT_NTF_IND_SEND_RSP

This subevent returns the result of calling `ble_gatts_ntf_ind_send` or `ble_gatts_ntf_ind_send_by_handle` to send a GATT notification or indication. The subevent data type is `ble_gatts_ntf_ind_send_rsp_t`, including the status of the sent data, service id, and attribute index.

● BLE_SRV_EVT_NTF_IND_MTP_SEND_RSP

This subevent returns the result of calling `ble_gatts_ntf_ind_mtp_send` to send notifications or indications to multiple remote devices. The message data type is `ble_gatts_ntf_ind_mtp_send_rsp_t`, including the status of the sent data, service id, and attribute index.

2.10.2. `ble_gatts_init`

Prototype: `ble_status_t ble_gatts_init(void)`

Function: Initialize the BLE GATT server module

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.10.3. `ble_gatts_svc_add`

Prototype: `ble_status_t ble_gatts_svc_add(uint8_t *p_svc_id, const uint8_t *uuid, uint16_t start_hdl, uint8_t info, const void *p_table, uint16_t table_length, p_fun_srv_cb srv_cb)`

Function: Add a service to the GATT server module.

Input parameter: uuid, service UUID address

`start_hdl`, service start attribute handle value; 0 means that the handle is not specified and is automatically assigned by the module

`info`, service information. For details, see `ble_gatt_svc_info_bf`

`p_table`, all attribute arrays of the service; each attribute

structure is `ble_gatt_attr_desc_t`

`table_length`, the length of service attribute array

`srv_cb`, the message handler function of GATT server. For the message type, see [gatts message type](#)

Output parameter: `p_svc_id`, the ID assigned by the BLE GATT server module to the service

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

After execution, a `BLE_PRF_MGR_EVT_SVC_ADD_RSP` message is sent to the callback function

2.10.4. **ble_gatts_svc_rmv**

Prototype: ble_status_t ble_gatts_svc_rmv(uint8_t svc_id)

Function: remove a service

Input parameter: svc_id, the ID assigned to the service when ble_gatts_svc_add is called

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After execution, a BLE_SRV_EVT_SVC_RMV_RSP message is sent to the callback function

2.10.5. **ble_gatts_ntf_ind_send**

Prototype: ble_status_t ble_gatts_ntf_ind_send (uint8_t conn_idx, uint8_t svc_id, uint16_t att_idx, uint8_t *p_val, uint16_t len, ble_gatt_evt_type_t evt_type)

Function: Send a notification/indication

Input parameter: conn_idx, connection index

svc_id, the ID assigned to the service when ble_gatts_svc_add is called

att_idx, the index value of the attribute in the array when

ble_gatts_svc_add is called

p_val, the address of data to be sent

len, the length of data to be sent

evt_type, whether the type of data sent this time is notification or indication

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

After execution, a BLE_SRV_EVT_GATT_OPERATION message with a subevent of BLE_SRV_EVT_NTF_IND_SEND_RSP is sent to the callback function

2.10.6. **ble_gatts_ntf_ind_send_by_handle**

Prototype: ble_status_t ble_gatts_ntf_ind_send_by_handle(uint8_t conn_idx, uint16_t handle, uint8_t *p_val, uint16_t len, ble_gatt_evt_type_t evt_type)

Function: Send a notification/indication through the attribute handle

Input parameter: `conn_idx`, connection index

`handle`, the handle value of the attribute, which can be obtained through

the index of the attribute in the array and the start handle of

the service when `ble_gatts_svc_add` is called

`p_val`, the address of data to be sent

`len`, the length of data to be sent

`evt_type`, whether the type of data sent this time is notification or indication

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

After execution, a `BLE_SRV_EVT_GATT_OPERATION`

message with the subevent of `BLE_SRV_EVT_NTF_IND_SEND_RSP`

is sent to the callback function

2.10.7. `ble_gatts_ntf_ind_mtp_send`

Prototype: `ble_status_t ble_gatts_ntf_ind_mtp_send (uint32_t conidx_bf, uint8_t svc_id, uint16_t att_idx, uint8_t *p_val, uint16_t len, ble_gatt_evt_type_t evt_type)`

Function: Send a notification/indication to multiple connections

Input parameter: `conidx_bf`, connection index bit combination, bit 0 represents

connection index 0x00, bit 1 represents connection index 0x01, and so on

`svc_id`, the ID assigned to the service when `ble_gatts_svc_add` is called

`att_idx`, the index value of the attribute in the array when

`ble_gatts_svc_add` is called

`p_val`, the address of data to be sent

`len`, the length of data to be sent

`evt_type`, whether the type of data sent this time is notification or indication

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

After execution, a `BLE_SRV_EVT_GATT_OPERATION`

message with the subevent of `BLE_SRV_EVT_NTF_IND_MTP_SEND_RSP`

is sent to the callback function

2.10.8. **ble_gatts_mtu_get**

Prototype: `ble_status_t ble_gatts_mtu_get(uint8_t conidx, uint16_t *p_mtu)`

Function: Obtain GATT MTU of the connection.

Input parameter: `conn_idx`, connection index

Output parameter: `p_mtu`, obtained GATT MTU of the connection

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.10.9. **ble_gatts_svc_attr_write_cfm**

Prototype: `ble_status_t ble_gatts_svc_attr_write_cfm(uint8_t conn_idx, uint16_t token, uint16_t status)`

Function: When receiving `BLE_SRV_EVT_GATT_OPERATION` and the subevent is `BLE_SRV_EVT_WRITE_REQ`, if automatic reply by GATT server module is not needed, `pending_cfm` in the message data should be sent to true, then `ble_gatts_svc_attr_write_cfm` should be called by user to confirm the write request according to user requirement.

Input parameter: `conn_idx`, connection index

`token`, GATT token, which is obtained in the

`BLE_SRV_EVT_WRITE_REQ` message

`status`, a status of replying to the write request

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.10.10. **ble_gatts_svc_attr_read_cfm**

Prototype: `ble_status_t ble_gatts_svc_attr_read_cfm(uint8_t conn_idx, uint16_t token, uint16_t status, uint16_t total_len, uint16_t value_len, uint8_t *p_value)`

Function: When receiving `BLE_SRV_EVT_GATT_OPERATION` and the subevent is `BLE_SRV_EVT_READ_REQ`, if automatic reply by GATT server module is not needed, `pending_cfm` in the message data should be sent to true,

then `ble_gatts_svc_attr_read_cfm` should be called by user to confirm the read request according to user requirement.

Input parameter: `conn_idx`, connection index

`token`, GATT token, which is obtained in the

`BLE_SRV_EVT_READ_REQ` message

`status`, a status of replying to the read request

`total_len`, the total length of attribute to be read

`value_len`, the attribute data length replied with to the read request

`p_value`, the attribute data content replied with to the read request

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.10.11. `ble_gatts_get_start_hdl`

Prototype: `ble_status_t ble_gatts_get_start_hdl(uint8_t svc_id, uint16_t *p_handle)`

Function: Obtain the start handle value allocated by the GATT server module to the service.

Input parameter: `svc_id`, service id, which is obtained in `ble_gatts_svc_add`

Output parameter: `p_handle`, obtained start handle value

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.11. BLE gattc API

The header file is `ble_gattc.h`.

The BLE GATT client module mainly provides the following functions: GATT discovery; read and write attribute value from the peer GATT server, etc.

2.11.1. gattc message type

BLE services can register a callback function to the BLE GATT client module, which will send the following event messages to BLE services through the callback function.

■ `BLE_CLI_EVT_CONN_STATE_CHANGE_IND`

This message is sent to the callback function when the device connection state changes. The message data type is `ble_gattc_conn_state_change_ind_t`, including the connection state `conn_state`. If the connection state is connected, the connection index and address

information of the peer device will be included; however, if the connection state is disconnected, the reason for disconnection will also be included.

■ BLE_CLI_EVT_GATT_OPERATION

This message is sent to the callback function when interacting with the peer GATT server. The message data type is `ble_gattc_op_info_t`, including the subevent `gattc_op_sub_evt` of GATT client operation, connection index `conn_idx`, and message data of different subevents. The message includes the following subevents:

- BLE_CLI_EVT_SVC_DISC_DONE_RSP

After `ble_gattc_start_discovery` is called to discover services of the peer GATT server, this subevent returns whether the registered service is found. The subevent data type is `ble_gattc_svc_dis_done_t`, including whether the service is found and the number of instances.

- BLE_CLI_EVT_READ_RSP

This subevent returns the result of reading the data of peer GATT server attribute by calling `ble_gattc_read`. The subevent data type is `ble_gattc_read_rsp_t`, including service uuid and characteristic uuid.

- BLE_CLI_EVT_WRITE_RSP

This subevent returns the result of writing data to the peer GATT server by calling `ble_gattc_write_req`, `ble_gattc_write_cmd`, or `ble_gattc_write_signed`. The subevent data type is `ble_gattc_write_rsp_t`, including service uuid and characteristic uuid.

- BLE_CLI_EVT_NTF_IND_RCV

When the peer GATT server sends notification or indication, this subevent is sent to the registered callback function. The subevent data type is `ble_gattc_ntf_ind_t`, including service uuid, characteristic uuid, and attribute handle.

2.11.2. **ble_gattc_init**

Prototype: `ble_status_t ble_gattc_init(void)`

Function: Initialize the GATT client module.

Input parameter: None

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.11.3. **ble_gattc_start_discovery**

Prototype: `ble_status_t ble_gattc_start_discovery(uint8_t conn_idx,`

p_discovery_done_cb callback)

Function: Start to discover services in the peer GATT server.

Input parameter: conn_idx, connection index

callback, discovery complete callback function

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.11.4. ble_gattc_svc_reg

Prototype: ble_status_t ble_gattc_svc_reg(ble_uuid_t *p_svc_uuid, p_fun_cli_cb p_cb)

Function: Register the callback function and service UUID to the BLE GATT client module.

Input parameter: p_svc_uuid, service uuid client pays attention to

p_cb, the message handler function of GATT client. For the message type,

see [gattc message type](#).

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.11.5. ble_gattc_svc_unreg

Prototype: ble_status_t ble_gattc_svc_unreg(ble_uuid_t *p_svc_uuid)

Function: Unregister the callback function and service UUID from BLE GATT client module

Input parameter: p_svc_uuid, service uuid to be unregistered

Output parameter: None

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.11.6. ble_gattc_read

Prototype: ble_status_t ble_gattc_read(uint8_t conidx, uint16_t hdl, uint16_t offset,
uint16_t length)

Function: Read the attribute data of the peer GATT server.

Input parameter: conidx, connection index

hdl, attribute handle

offset, data offset to be read

length, data length to be read

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure After execution, the `BLE_CLI_EVT_GATT_OPERATION` message with a subevent of `BLE_CLI_EVT_READ_RSP` will be sent to the registered callback function.

2.11.7. **ble_gattc_write_req**

Prototype: `ble_status_t ble_gattc_write_req(uint8_t conidx, uint16_t hdl, uint16_t length, uint8_t *p_value)`

Function: Write data to peer server (write request)

Input parameter: conidx, connection index

hdl, attribute handle

length, data length to be written

p_value, data to be written

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure After execution, the `BLE_CLI_EVT_GATT_OPERATION` message with a subevent of `BLE_CLI_EVT_WRITE_RSP` will be sent to the callback function.

2.11.8. **ble_gattc_write_cmd**

Prototype: `ble_status_t ble_gattc_write_cmd(uint8_t conidx, uint16_t hdl, uint16_t length, uint8_t *p_value)`

Function: Write data to peer server (write command)

Input parameter: conidx, connection index

hdl, attribute handle

length, data length to be written

p_value, data to be written

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure. After execution, the `BLE_CLI_EVT_GATT_OPERATION` message with a subevent of `BLE_CLI_EVT_WRITE_RSP` will be sent to the registered callback function.

2.11.9. `ble_gattc_write_signed`

Prototype: `ble_status_t ble_gattc_write_cmd(uint8_t conidx, uint16_t hdl, uint16_t length, uint8_t *p_value)`

Function: Write data to peer server (write signed)

Input parameter: `conidx`, connection index

`hdl`, attribute handle

`length`, data length to be written

`p_value`, data to be written

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure. After execution, the `BLE_CLI_EVT_GATT_OPERATION` message with a subevent of `BLE_CLI_EVT_WRITE_RSP` will be sent to the registered callback function.

2.11.10. `ble_gattc_mtu_update`

Prototype: `ble_status_t ble_gattc_mtu_update(uint8_t conidx)`

Function: Update GATT mtu.

Input parameter: `conidx`, connection index

Output parameter: None

Return value: Return 0 on success, and return the error code defined in `ble_status_t` on failure

2.11.11. `ble_gattc_mtu_get`

Prototype: `ble_status_t ble_status_t ble_gattc_mtu_get(uint8_t conidx, uint16_t *p_mtu)`

Function: Obtain the GATT mtu value of the connection.

Input parameter: `conidx`, connection index

Output parameter: p_mtu, mtu size

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.11.12. ble_gattc_find_char_handle

Prototype: ble_status_t ble_gattc_find_char_handle(uint8_t conn_idx, ble_gattc_uuid_info_t
*svc_uuid, ble_gattc_uuid_info_t *char_uuid, uint16_t *handle)

Function: Find the value handle value of the characteristic.

Input parameter: conidx, connection index

svc_uuid, service uuid

char_uuid, characteristic uuid

Output parameter: handle, attribute handle value

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.11.13. ble_gattc_find_desc_handle

Prototype: ble_status_t ble_gattc_find_desc_handle(uint8_t conn_idx, ble_gattc_uuid_info_t
*svc_uuid, ble_gattc_uuid_info_t *char_uuid,
ble_gattc_uuid_info_t *desc_uuid, uint16_t *handle)

Function: Find the handle value of description.

Input parameter: conidx, connection index

svc_uuid, service uuid

char_uuid, characteristic uuid

desc_uuid, description uuid

Output parameter: handle, attribute handle value

Return value: Return 0 on success, and return the error code defined in ble_status_t on failure

2.12. BLE export API

The header file is ble_export.h.

The file contains the initialization of the BLE stack, BLE task and BLE APP task.

2.12.1. ble_stack_init

Prototype: void ble_stack_init(ble_cfg_func_t en_cfg, ble_os_api_t *p_os_api)

Function: Initialize the BLE stack.

Input parameter: `en_cfg`, for details about the configurations to be enabled, see

`ble_cfg_func_t`

`p_os_api`, OS related api functions

Output parameter: None

Return value: None

2.12.2. **ble_stack_task_resume**

Prototype: `void ble_stack_task_resume(bool isr)`

Function: Resume the BLE stack task. If BLE task is in the sleep mode, this function can be called by an external interrupt to wake up the BLE task.

Input parameter: `isr`, which indicates whether it is called by an interrupt

Output parameter: None

Return value: None

2.12.3. **ble_stack_task_init**

Prototype: `uint32_t ble_stack_task_init(uint32_t stack_size, uint32_t priority)`

Function: Initialize the BLE task.

Input parameter: `stack_size`, the size of the task stack in four bytes

`priority`, task priority

Output parameter: None

Return value: 0 upon successful execution and other values upon failure.

2.12.4. **ble_stack_task_deinit**

Prototype: `void ble_stack_task_deinit(void)`

Function: Delete the BLE task.

Input parameter: None

Output parameter: None

Return value: None

2.12.5. **ble_app_task_init**

Prototype: `uint32_t ble_app_task_init(uint32_t stack_size, uint32_t priority)`

Function: Initialize the BLE APP task.

Input parameter: `stack_size`, the size of the task stack in four bytes
`priority`, task priority

Output parameter: None

Return value: 0 upon successful execution and other values upon failure.

2.12.6. **ble_app_task_deinit**

Prototype: `void ble_app_task_deinit(void)`

Function: Delete the BLE APP task.

Input parameter: None

Output parameter: None

Return value: None

2.12.7. **ble_local_app_msg_send**

Prototype: `bool ble_local_app_msg_send (void *p_msg, uint16_t msg_len)`

Function: If the upper layer determines to handle the message asynchronously, it can send a message to the BLE APP task, specifying that the message should be handled in the callback function. In this case, `ble_app_msg_hdl_reg` should be called to register the callback function in advance.

Input parameter: `p_msg`, message content
`msg_len`, the length of message content

Output parameter: None

Return value: Return "true" upon success and "false" upon failure.

2.12.8. **ble_app_msg_hdl_reg**

Prototype: `void ble_app_msg_hdl_reg(ble_app_msg_hdl_t p_hdl)`

Function: Work together with `ble_local_app_msg_send` to register the callback function for APP message.

Input parameter: p_hdl, callback function

Output parameter: None

Return value: None

2.12.9. **ble_sleep_mode_set**

Prototype: void ble_sleep_mode_set(uint8_t mode)

Function: Set the BLE sleep mode.

Input parameter: mode: 0 means normal mode, while 1 means sleep mode. (If there are no task to deal with, the task and BLE core are in the sleep mode)

Output parameter: None

Return value: None

2.12.10. **ble_sleep_mode_get**

Prototype: uint8_t ble_sleep_mode_get(void)

Function: Get the BLE sleep mode.

Input parameter: None

Output parameter: None

Return value: mode: 0 means normal mode, while 1 means sleep mode. (If there are no tasks, the task and BLE core are in the sleep mode)

2.12.11. **ble_core_is_deep_sleep**

Prototype: bool ble_core_is_deep_sleep(void)

Function: Query whether BLE core is in the deep sleep mode.

Input parameter: None

Output parameter: None

Return value: true for the deep sleep mode and false for other modes

2.12.12. **ble_modem_config**

Prototype: void ble_modem_config(void)

Function: Configure the modem parameter under BLE core every time it is woken up from the sleep mode.

Input parameter: None

Output parameter: None

Return value: None

2.12.13. **ble_work_status_set**

Prototype: void ble_work_status_set(enum ble_work_status_t mode)

Function: Set the working status of BLE, through which it can be dynamically enabled

and disable with reference to ble_enable and ble_disable among basic commands.

Input parameter: mode: 0 means enable, while 1 means disable.

Output parameter: None

Return value: None

2.12.14. **ble_work_status_get**

Prototype: ble_work_status_t ble_work_status_get(void)

Function: Get the BLE working status.

Input parameter: None

Output parameter: None

Return value: mode: 0 means enable, while 1 means disable.

2.12.15. **ble_internal_encode**

Prototype: void ble_internal_encode(uint8_t *data, uint16_t len, uint8_t rand)

Function: Encode the data by using the internal algorithm.

Input parameter: data, input data

len, the length of input data

rand, random number, through which different values can be output from

the same input

Output parameter: data, encoded data

Return value: None

2.12.16. **ble_internal_decode**

Prototype: void ble_internal_decode(uint8_t *data, uint16_t len, uint8_t rand)

Function: Decode the data by using the internal algorithm.

Input parameter: data, input data

len, the length of input data

rand, random number, through which different values can be output from
the same input

Output parameter: data, decoded data

Return value: None

2.12.17. **ble_register_hci_uart**

Prototype: bool ble_register_hci_uart(ble_uart_func_t *p_func)

Function: Register HCI uart related callback functions for the special lib of controller only.

Input parameter: p_func, for the callback function of the uart, see ble_uart_func_t

Output parameter: None

Return value: True indicates that the registration is successful; otherwise, false

3. Application examples

3.1. Scan

The BLE scan function is used to find Bluetooth low energy devices in the surrounding environment. Enabling the scan function will report the scanned devices to the application layer.

Quickly use the function in the following steps:

1. Register an event handler to handle changes in scan status and report advertising data.

Table 3-1. Example code of scan event handler

```
static void ble_app_scan_mgr_evt_handler(ble_scan_evt_t event, ble_scan_data_u *p_data)
{
    switch (event) {
        case BLE_SCAN_EVT_STATE_CHG:
            if (p_data->scan_state.scan_state == BLE_SCAN_STATE_ENABLED) {
                dbg_print(NOTICE, "Ble Scan enabled status 0x%x\r\n", p_data->scan_state.reason);
            } else if (p_data->scan_state.scan_state == BLE_SCAN_STATE_ENABLING) {
                scan_mgr_clear_dev_list();
            } else if (p_data->scan_state.scan_state == BLE_SCAN_STATE_DISABLED) {
                dbg_print(NOTICE, "Ble Scan disabled status 0x%x\r\n",
                p_data->scan_state.reason);
            }
            break;

        case BLE_SCAN_EVT_ADV_RPT:
            scan_mgr_report_hdlr(p_data->p_adv_rpt);
            break;
    }
}
```

2. Configure scan parameters through `ble_scan_param_set`. The structure parameters are as follows:

`type`---scan type, which can be set to *general discovery (general scan)*, *limit discovery (limit scan)*, etc.

`prop`---scan attribute, which can be set to *active scan* or *passive scan of 1M and CODED PHY*, *filter strategy*, etc.

`dup_filt_pol`---duplicate filtering. When it is enabled, the received advertising signal will not be repeatedly reported to the application.

`scan_intv`---scan interval, how often the controller scans.

scan_win---scan window, the duration of each scan.

duration---scan duration, which indicates continuous scan when configured to 0.

period---whether to scan periodically, with the duration as the period.

Table 3-2. Example code of configure scan parameters

```

/**@brief Function for set scan parameters.
 *
 * @param[in] param          scan parameters (see enum #ble_gap_scan_param_t)
 * @retval BLE_ERR_NO_ERROR  If ble scan module disable successfully.
 */
ble_status_t ble_scan_param_set(ble_gap_scan_param_t *p_param);

/** The default scan parameters are as follows*/
p_ble_scan_env->param.type = BLE_GAP_SCAN_TYPE_GEN_DISC;
p_ble_scan_env->param.prop = BLE_GAP_SCAN_PROP_PHY_1M_BIT |
                             BLE_GAP_SCAN_PROP_ACTIVE_1M_BIT |
                             BLE_GAP_SCAN_PROP_PHY_CODED_BIT |
                             BLE_GAP_SCAN_PROP_ACTIVE_CODED_BIT;
p_ble_scan_env->param.dup_filt_pol = BLE_GAP_DUP_FILTER_EN;
p_ble_scan_env->param.scan_intv_1m = 160; // 100ms
p_ble_scan_env->param.scan_intv_coded = 160; // 100ms
p_ble_scan_env->param.scan_win_1m = 48; // 30ms
p_ble_scan_env->param.scan_win_coded = 48; // 30ms
p_ble_scan_env->param.duration = 0;
p_ble_scan_env->param.period = 0;

```

3. To enable the scan function, call ble_scan_enable API.

Table 3-3. Example code of enable scan

```

void app_scan_enable(bool update_rssi)
{
    if (ble_scan_enable() != BLE_ERR_NO_ERROR) {
        dbg_print(NOTICE, "app_scan_enable fail!\n");
        return;
    }
}

```

3.2. Advertising

The BLE advertising function is used to send advertising messages, allowing surrounding BLE devices to discover and connect it or send periodic data, etc. It can be configured as legacy advertising (traditional advertising), extended advertising, and periodic advertising.

Quickly use the function in the following steps:

1. Register an event handler to handle changes in advertising status and report received scan requests.

Table 3-4. Example code of advertising event handler

```

static void app_adv_mgr_evt_hdlr(ble_adv_evt_t adv_evt, void *p_data, void *p_context)
{
    app_adv_actv_t *p_adv = (app_adv_actv_t *)p_context;

    switch (adv_evt) {
    case BLE_ADV_EVT_STATE_CHG: {
        ble_adv_state_chg_t *p_chg = (ble_adv_state_chg_t *)p_data;
        ble_adv_state_t old_state = p_adv->state;

        dbg_print(NOTICE, "adv state change 0x%x ==> 0x%x, reason 0x%x\r\n", old_state,
p_chg->state, p_chg->reason);

        p_adv->state = p_chg->state;

        if ((p_chg->state == BLE_ADV_STATE_CREATE) && (old_state ==
BLE_ADV_STATE_CREATING)) {
            p_adv->idx = p_chg->adv_idx;
            app_print("adv index %d\r\n", p_adv->idx);

            app_adv_start(p_adv);
        } else if ((p_chg->state == BLE_ADV_STATE_CREATE) && (old_state ==
BLE_ADV_STATE_START)) {
            dbg_print(NOTICE, "adv stopped, remove %d\r\n", p_adv->remove_after_stop);

            if (p_adv->remove_after_stop) {
                ble_adv_remove(p_adv->idx);
                p_adv->remove_after_stop = false;
            }
        } else if (p_chg->state == BLE_ADV_STATE_IDLE) {
            free_adv_actv(p_adv);
        }
    } break;

    case BLE_ADV_EVT_DATA_UPDATE_RSP: {
        ble_adv_data_update_rsp_t *p_rsp = (ble_adv_data_update_rsp_t *)p_data;
        dbg_print(NOTICE, "adv data update rsp, type %d, status 0x%x\r\n", p_rsp->type,
p_rsp->status);
    } break;

    case BLE_ADV_EVT_SCAN_REQ_RCV: {

```

```

ble_adv_scan_req_rcv_t *p_req = (ble_adv_scan_req_rcv_t *)p_data;
dbg_print(NOTICE, "scan req rcv, device addr %02X:%02X:%02X:%02X:%02X:%02X\r\n",
          p_req->peer_addr.addr[5], p_req->peer_addr.addr[4], p_req->peer_addr.addr[3],
          p_req->peer_addr.addr[2], p_req->peer_addr.addr[1], p_req->peer_addr.addr[0]);
} break;

default:
    break;
}
}

```

- The device sends a advertising message mainly in two steps: create a advertising and enable it. The advertising can be enabled only in successfully created status. For example, the following application layer creates advertising code and configures different advertising parameters based on different advertising types.

Table 3-5. Example code of create advertising

```

ble_status_t app_adv_create(app_adv_param_t *p_param)
{
    app_adv_actv_t *p_adv;
    ble_adv_param_t adv_param = {0};

    p_adv = get_free_adv_actv();
    if (p_adv == NULL) {
        return BLE_ERR_NO_RESOURCES;
    }

    p_adv->max_data_len = p_param->max_data_len;

    adv_param.param.own_addr_type = p_param->own_addr_type;

    if (p_param->type == BLE_ADV_TYPE_LEGACY) {
        adv_param.param.type = BLE_GAP_ADV_TYPE_LEGACY;
        adv_param.param.prop = p_param->prop;

        if (p_param->wl_enable) {
            adv_param.param.filter_pol = BLE_GAP_ADV_ALLOW_SCAN_FAL_CON_FAL;
            adv_param.param.disc_mode = BLE_GAP_ADV_MODE_NON_DISC;
        } else {
            adv_param.param.filter_pol = BLE_GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;
            adv_param.param.disc_mode = p_param->disc_mode;
        }

        adv_param.param.ch_map = APP_ADV_CHMAP;
        adv_param.param.primary_phy = p_param->pri_phy;
    }
}

```

```

} else if (p_param->type == BLE_ADV_TYPE_EXTENDED) {
    adv_param.param.type = BLE_GAP_ADV_TYPE_EXTENDED;
    adv_param.param.prop = p_param->prop;

    if (p_param->wl_enable) {
        adv_param.param.filter_pol = BLE_GAP_ADV_ALLOW_SCAN_FAL_CON_FAL;
        adv_param.param.disc_mode = BLE_GAP_ADV_MODE_NON_DISC;
    } else {
        adv_param.param.filter_pol = BLE_GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;
        adv_param.param.disc_mode = p_param->disc_mode;
    }

    adv_param.param.ch_map = APP_ADV_CHMAP;
    adv_param.param.primary_phy = p_param->pri_phy;
    adv_param.param.adv_sid = get_adv_sid();
    adv_param.param.max_skip = 0x00;
    adv_param.param.secondary_phy = p_param->sec_phy;
} else {
    return BLE_GAP_ERR_INVALID_PARAM;
}

if (adv_param.param.prop & BLE_GAP_ADV_PROP_DIRECTED_BIT) {
    adv_param.param.peer_addr = p_param->peer_addr;
    adv_param.param.disc_mode = BLE_GAP_ADV_MODE_NON_DISC;
    p_adv->peer_addr = p_param->peer_addr;
}

if (adv_param.param.prop & BLE_GAP_ADV_PROP_ANONYMOUS_BIT) {
    adv_param.param.disc_mode = BLE_GAP_ADV_MODE_NON_DISC;
}

p_adv->disc_mode = adv_param.param.disc_mode;

adv_param.param.adv_intv_min = APP_ADV_INT_MIN;
adv_param.param.adv_intv_max = APP_ADV_INT_MAX;

if (p_adv->disc_mode == BLE_GAP_ADV_MODE_LIM_DISC) {
    adv_param.param.duration = 1000;    // 10s
}

if (p_param->type != BLE_ADV_TYPE_LEGACY) {
    adv_param.include_tx_pwr = true;
    adv_param.scan_req_ntf = true;
}

```

```

}

return ble_adv_create(&adv_param, app_adv_mgr_evt_hdlr, p_adv);

}

```

3. Enable the advertising. After receiving the message that the advertising is successfully created in the registered event handler, call the `ble_adv_start` interface to enable the advertising. Afterwards, receiving the reported advertising status `BLE_ADV_STATE_START` in the event handler means that the advertising is enabled successfully.

The last three parameters in the `ble_adv_start` API are used to set advertising data, scan response data, and periodic advertising data respectively. The content can be set directly by the application layer or packaged by the BLE ADV module through configuration parameters. For example, all data are set directly by the application layer in the following code.

Table 3-6. Example code of enable advertising

```

static uint8_t adv_data_1[7] = {0x06, 0x16, 0x52, 0x18, 0x18, 0x36, 0x9A};
static uint8_t per_data_1[52] = {0x33, 0x16, 0x51, 0x18, 0x40, 0x9c, 0x00, 0x01, 0x02, 0x06,
                                0x00, 0x00, 0x00, 0x00, 0x0d, 0x02, 0x01, 0x08, 0x02, 0x02,
                                0x01, 0x03, 0x04, 0x78, 0x00, 0x02, 0x05, 0x01, 0x07, 0x03,
                                0x02, 0x04, 0x00, 0x02, 0x04, 0x80, 0x01, 0x06, 0x05, 0x03,
                                0x00, 0x04, 0x00, 0x00, 0x02, 0x06, 0x05, 0x03, 0x00, 0x08,
                                0x00, 0x00
                                };

static void app_adv_start(app_adv_actv_t *p_adv)
{
    ble_adv_data_set_t adv;
    ble_adv_data_set_t scan_rsp;
    ble_adv_data_set_t per_adv;
    ble_data_t adv_data;
    ble_data_t per_adv_data;

    adv.data_force = true;
    scan_rsp.data_force = true;
    per_adv.data_force = true;

    adv_data.len = 7;
    adv_data.p_data = adv_data_1;

    per_adv_data.len = 52;
    per_adv_data.p_data = per_data_1;

    adv.data.p_data_force = &adv_data;
}

```

```

scan_rsp.data.p_data_force = &adv_data;
per_adv.data.p_data_force = &per_adv_data;

ble_adv_start(p_adv->idx, &adv, &scan_rsp, &per_adv);
}

```

3.3. GATT server application

GD32VW553 SDK provides functions such as adding/deleting services and sending notification/indication as BLE GATT server role. Users can implement specific services according to their requirements. For specific APIs, see [BLE gatts API](#) 错误!未找到引用源。. Here is an example of DIS to illustrate how to use these APIs to implement a service server. The file is MSDK\ble\profile\dis\ble_diss.c.

3.3.1. Adding a service

Add a service to the BLE GATT server module through the `ble_gatts_svc_add` function, whose input parameters include service UUID, service attribute database, callback handler for GATT server message, etc. Service UUID can be 16-bit, 32-bit, or 128-bit. They need to be described in the *info and table type* parameters. For example, in the code of ble diss, UUID 16 is used for service UUID. When calling the `ble_gatts_svc_add` function, use `SVC_UUID(16)` to describe it.

Table 3-7. Example code of add a service

```

ret = ble_gatts_svc_add(&ble_diss_svc_id, ble_dis_uuid, 0, SVC_UUID(16), ble_diss_attr_db,
BLE_DIS_HDL_NB, ble_diss_srv_cb);

```

3.3.2. Service attribute database

Service attribute database is an array composed of a series of `ble_gatt_attr_desc_t` elements. Each element in the array is an attribute, which can be primary service, characteristic declaration, characteristic value declaration, etc. Users can freely combine them according to the requirements of different services.

Each attribute consists of a UUID and its attribute description. All attributes in DIS are read-only, so just specify the RD property. For the characteristic value declaration, the maximum size of the value can also be specified.

Table 3-8. Example code of service database

```

const ble_gatt_attr_desc_t ble_diss_attr_db[BLE_DIS_HDL_NB] =
{
    [BLE_DIS_HDL_SVC] = {UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_PRIMARY_SERVICE),
PROP(RD), 0},

```

```

[BLE_DIS_HDL_MANUFACT_NAME_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_MANUFACT_NAME_VAL] =
{UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_MANUF_NAME), PROP(RD),
BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_MODEL_NB_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_MODEL_NB_VAL] =
{UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_MODEL_NB), PROP(RD),
BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_SERIAL_NB_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_SERIAL_NB_VAL] =
{UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_SERIAL_NB), PROP(RD),
BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_HARD_REV_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_HARD_REV_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_HW_REV),
PROP(RD), BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_FIRM_REV_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_FIRM_REV_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_FW_REV),
PROP(RD), BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_SW_REV_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_SW_REV_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_SW_REV),
PROP(RD), BLE_DIS_VAL_MAX_LEN},

[BLE_DIS_HDL_SYSTEM_ID_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_SYSTEM_ID_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_SYS_ID),
PROP(RD), BLE_DIS_SYS_ID_LEN},

[BLE_DIS_HDL_IEEE_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
[BLE_DIS_HDL_IEEE_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_IEEE_CERTIF),
PROP(RD), BLE_DIS_VAL_MAX_LEN},

```

```
[BLE_DIS_HDL_PNP_ID_CHAR] =
{UUID_16BIT_TO_ARRAY(BLE_GATT_DECL_CHARACTERISTIC), PROP(RD), 0},
  [BLE_DIS_HDL_PNP_ID_VAL] = {UUID_16BIT_TO_ARRAY(BLE_DIS_CHAR_PNP_ID),
PROP(RD), BLE_DIS_PNP_ID_LEN},
};
```

3.3.3. Service attribute read and write

The last parameter of `ble_gatts_svc_add` is to register a GATT server event handler callback function, which is executed when the peer client performs read or write operation on the service, GATT server event type is `BLE_SRV_EVT_GATT_OPERATION`, subevent type is `BLE_SRV_EVT_READ_REQ` or `BLE_SRV_EVT_WRITE_REQ`, subevent data structure is `ble_gatts_read_req_t` or `ble_gatts_write_req_t`, in which there is an `att_idx` parameter indicates the corresponding attribute index in database table when registered.

Table 3-9. Example code of attribute read and write function

```
ble_status_t ble_diss_srv_cb(ble_gatts_msg_info_t *p_srv_msg_info)
{
    uint8_t attr_idx = 0;
    uint16_t len = 0;
    uint8_t attr_len = 0;
    uint8_t *p_attr = NULL;

    if (p_srv_msg_info->srv_msg_type == BLE_SRV_EVT_GATT_OPERATION) {
        if (p_srv_msg_info->msg_data.gatts_op_info.gatts_op_sub_evt ==
BLE_SRV_EVT_READ_REQ) {
            ble_gatts_read_req_t * p_read_req =
&p_srv_msg_info->msg_data.gatts_op_info.gatts_op_data.read_req;

            attr_idx = p_read_req->att_idx;
            switch (attr_idx) {
                case BLE_DIS_HDL_MANUFACT_NAME_VAL: {
                    p_attr = ble_diss_val.manufact_name;
                    attr_len = ble_diss_val.manufact_name_len;
                } break;

                case BLE_DIS_HDL_MODEL_NB_VAL: {
                    p_attr = ble_diss_val.model_num;
                    attr_len = ble_diss_val.model_num_len;
                } break;

                case BLE_DIS_HDL_SERIAL_NB_VAL: {
                    p_attr = ble_diss_val.serial_num;
                    attr_len = ble_diss_val.serial_num_len;
```

```
    } break;

    case BLE_DIS_HDL_HARD_REV_VAL: {
        p_attr    = ble_diss_val.hw_rev;
        attr_len = ble_diss_val.hw_rev_len;
    } break;

    case BLE_DIS_HDL_FIRM_REV_VAL: {
        p_attr    = ble_diss_val.fw_rev;
        attr_len = ble_diss_val.fw_rev_len;
    } break;

    case BLE_DIS_HDL_SW_REV_VAL: {
        p_attr    = ble_diss_val.sw_rev;
        attr_len = ble_diss_val.sw_rev_len;
    } break;

    case BLE_DIS_HDL_SYSTEM_ID_VAL: {
        p_attr    = ble_diss_val.sys_id;
        attr_len = BLE_DIS_SYS_ID_LEN;
    } break;

    case BLE_DIS_HDL_IEEE_VAL: {
        p_attr    = ble_diss_val.ieee_data;
        attr_len = ble_diss_val.ieee_data_len;
    } break;

    case BLE_DIS_HDL_PNP_ID_VAL: {
        p_attr    = ble_diss_val.pnp_id;
        attr_len = BLE_DIS_PNP_ID_LEN;
    } break;

    default:
        return BLE_ATT_ERR_INVALID_HANDLE;
    }

    if (p_read_req->offset > attr_len) {
        return BLE_ATT_ERR_INVALID_OFFSET;
    }

    len = ble_min(p_read_req->max_len, attr_len - p_read_req->offset);
    p_read_req->val_len = len;
    memcpy(p_read_req->p_val, p_attr, len);
```



```

    }
}

return BLE_ERR_NO_ERROR;

```

If there is an attribute in the service that supports Client Characteristic Configuration declaration (CCCD) and if peer client enables it, then `ble_srv_ntf_ind_send` interface can be used to send a notification/indication.

Table 3-10. Example code of send notification

```

static void bcwl_ntf_event_send(uint8_t *p_val, uint16_t len)
{
    if (bcwl_env.ntf_cfg == 0) {
        dbg_print(ERR, "%s fail\r\n", __func__);
        return;
    }

    ble_gatts_ntf_ind_send(bcwl_env.conn_id, bcwl_env.prf_id, BCW_IDX_NTF, p_val, len,
        BLE_GATT_NOTIFY);
}

```

3.4. BLE distribution network

Blue courier is a BLE-based WiFi network configuration function. The SSID, password, channel, and encryption type of WiFi are transferred through the protocol to the GD device, which can be connected to AP through such information or establish SoftAP. The link supports data fragmentation and CRC16 integrity verification. Security relies on the encryption of the BLE link, and the encoding method is taken for the transfer of the message containing SSID and password to avoid transferring the plaintext over the air. Execute the `ble_courier_wifi` command in the "AN153 GD32VW553 Basic Commands User Guide".

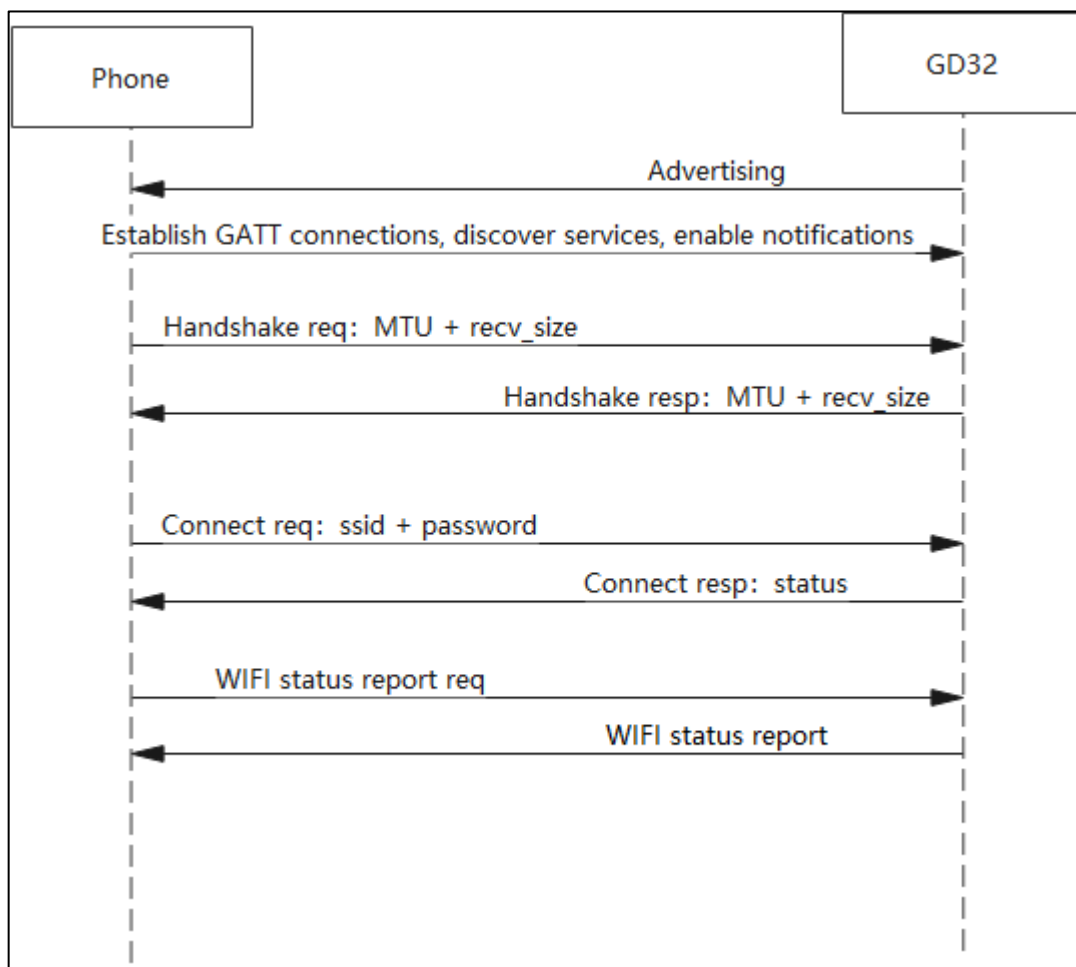
3.4.1. Process of Blue courier

By taking the example of configuring WiFi as the station to connect to AP, the following introduces key steps of advertising, connection, service discovery, enable notification, handshake, data transfer, and reportconnection sate.

1. After Blue courier wifi is enabled, the GD device will register the service with the GATT server module and send the advertisingwith special advertising data. The advertising can be defined by the user as required.
2. After the advertising is searched for through a WeChat Mini Program, the phone as GATT Client will connect to the GD device.
3. After establishing the GATT connection, the phone will send the handshake request message to the GD device, which will return the handshake response message upon receiving the message.

4. The phone can send the following messages to the GD device: Connect to WiFi; create SoftAp; get the WiFi status.

Figure 3-1. Process of Blue courier



3.4.2. GATT description

To add a distribution network service, refer to the description in [Adding a service](#).

For the description of UUID used by the distribution network service, see [Table 3-11. Distribution network service UUID](#) 错误未找到引用源。.

Table 3-11. Distribution network service UUID

Attribute	Description
Blue courier WIFI Service	UUID = 0000FFF0-0000-1000-8000-00805F9B34FB
C1 Characteristic (Client TX Buffer)	UUID = 0000FFF1-0000-1000-8000-00805F9B34FB Characteristic Properties = Write max length = 256 bytes Security level=unauth (the link must be encrypted, and pairing is required for the first time of connection)
C2 Characteristic	UUID = 0000FFF2-0000-1000-8000-00805F9B34FB

(Client RX Buffer)	Characteristic Properties = Notify max length = 256 bytes
--------------------	--

3.4.3. Advertising data

The Blue courier WiFi Service UUID must be included in the advertising data so that other devices can discover that the local device supports the BLE distribution network function. The peer device can filter by Service UUID when searching for BLE devices. For details, see the following table.

Table 3-12. Service UUID in advertising data

Byte	Value	Description
0	0x03	AD[0] Length == 3 bytes
1	0x03	AD[0] Type == 1 (Flags) Complete list of 16 bit service UUIDs.
2-3	0xFFFF0	16-bit Blue courier WIFI Service UUID

3.4.4. Frame format

The frame format for communication between the mobile app of Blue courier and the GD device is as follows:

Table 3-13. Frame format of blue courier

Field	Size (byte)
flag	1
sequence	1
opcode	1
data_len	1
data	#{data_len}
crc	2

flag

The frame control field occupies a byte, where each bit has a different meaning, as listed in the following table:

Table 3-14. Frame control field

Bit	Meaning
0x01	Begin: It means whether the frame is the first fragment. <ul style="list-style-type: none"> • 0: It means the frame is the remaining fragment. • 1: It means the frame is the first fragment. • The fragment is used to transfer long data. Only the first two bytes in the data field of the first packet of the fragmented packet show the total length of the data content and are used to indicate the memory size allocated for peer receiving, namely, data = total_len + data.
0x02	End: It means whether the frame is the last fragment.

	<ul style="list-style-type: none"> 0: It means the frame is not the last fragment. 1: It means the frame is the last fragment. <p>If both the Begin and End bits are set to 1, the packet is not fragmented.</p>
0x04	<p>ACK: It means whether the receiver should reply with ACK.</p> <ul style="list-style-type: none"> 0: It means the receiver unnecessarily replies with ACK. 1: It means the receiver should reply with ACK.
0x08~0x80	Reserved

sequence

Sequence control field. When a frame is sent, regardless of its type, its sequence will automatically increase by 1 to prevent replay attack. The sequence will be cleared after each re-connection.

opcode

The opcode field occupies a byte, divided into two parts: Type and Subtype. The Type occupies two higher bits, which indicate the frame is the management or data frame. The Subtype occupies six lower bits, which indicate the meaning of the management or data frame.

1. Management frame (binary system: 0x0 b'00).

Table 3-15. Content of management frame

Management frame	Meaning	Description	Content
0x0 (b'000000)	Handshake	Handshake is used to exchange the mtus at both ends and the maximum receiving length, determining the size of the fragmented packet and the total length of the largest report. The mtu, whichever is smaller, should be taken as the fragment size at both ends. recv_size is the maximum peer receiving length, which should be taken as the maximum sending length by the receiver.	The data field totally occupies four bytes, including two bytes for mtu and two bytes for recv_size. Phone -> GD device: mtu + recv_size GD device -> phone: mtu + recv_size
0x1 (b'000001)	ACK	The data field of the ACK frame uses the sequential value of the response frame.	The data field occupies a byte, using the same sequential value as that of the response frame.

0x2 (b'000100)	Error reporting	The data field is used to report an error to the peer device. The error code can be defined by the user.	status: 1byte
-------------------	-----------------	--	---------------

2. Data frame (binary system: 0x1 b'01).

Table 3-16. Content of data frame

Data frame	Meaning	Description	Remarks
0x0 (b'000000)	Send the user-defined data.	The data field is used to transfer the user-defined data to the peer device for test.	
0x1 (b'000001)	Get the information of the WiFi scan list.	The phone sends the message with a length of 0 to the GD device. Upon receiving the message, the GD device will trigger WiFi scan and send the scan information through the message to the phone.	GD device -> phone: Structure of each ssid: len+rss+mode+ssid len = 2byte(rssi+mode) + ssid length
0x2 (b'000010)	Send the connection request of the STA device.	Upon receiving the information of AP to be connected by the STA device, the GD device will trigger WiFi connection and send the connection result to the phone. The sent data should be randomly encoded to avoid generating the same code data.	Phone -> GD device: ssid_len + ssid + password_len + password + random GD device -> phone: status ssid_len, password_len, random, status: 1byte
0x3 (b'000011)	Send the disconnection request of the STA device.	The phone sends the message with a length of 0 to the GD device. Upon receiving the message, the GD device will trigger WiFi disconnection and send the status to the phone.	GD device -> phone: status status: 1byte
0x4 (b'000100)	Send the request of creating the SoftAP mode.	Upon receiving the information of AP to be created by the device, the GD device will trigger softAp creation and send the creation result to the	Phone -> GD device: ssid_len + ssid + password_len + password + channel + akm + hide + random GD device -> phone:

		phone. The sent data should be randomly encoded to avoid generating the same code data.	status ssid_len, password_len, channel, akm, hide, random, status: 1byte
0x5 (b'000101)	Send the request of stopping the SoftAP mode.	The phone sends the message with a length of 0 to the GD device. Upon receiving the message, the GD device will trigger softAp stopping and send the status to the phone.	GD device -> phone: status status: 1byte
0x6 (b'000110)	Get WiFi status.	The phone sends the message with a length of 0 to the GD device. After receiving the message, the GD device will report WiFi status to the phone.	It will notify the phone of the current device mode, connection status, SSID, and channel by reporting the WiFi connection status to the phone. For the message content structure, refer to the app implementation end.

crc

crc16 is used for integrity verification for communication through Blue courier by making a calculation based on four parts, namely sequence, opcode, data_len, and data.

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Nov.24.2023
1.1	Add API such as ble_conn_enable_central_feat, ble_svc_data_save, ble_svc_data_load and ble_register_hci_uart. Modify API such as ble_stack_init and ble_stack_task_init.	Feb.29.2024
1.2	Add APIs such as ble_adp_callback_unregister, ble_adp_disable, ble_scan_callback_unregister, ble_sec_callback_unregister, ble_list_callback_unregister, ble_per_sync_callback_unregister, ble_gattc_svc_unreg, ble_stack_task_deinit, ble_app_task_deinit. Modify APIs such as ble_stack_init and ble_stack_task_init. Delete API such as ble_stack_task_suspend.	Jul.10.2024

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which have been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.