EC200U-CN QuecOpen Bluetooth API Reference Manual

LTE Standard Module Series

Version: 1.0.0

Date: 2021-03-13

Status: Preliminary

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm.

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or email to support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

Copyright

About the Document

Revision History

Version	Date	Author	Description
-	2021-03-09	Ryan YI/ Chaos HUANG	Creation of the document
1.0.0	2021-03-13	Ryan YI/ Chaos HUANG	Preliminary

Contents

About the Document **3**
Contents **4**
Table Index **7**

1 Introduction **8**

2 BT APIs **9**
2.1. General BT APIs 9
2.1.1. ql_bt_start 9
2.1.2. ql_bt_stop 9
2.1.3. ql_bt_get_state 10
2.1.4. ql_bt_ble_set_localname 10
2.1.5. ql_bt_ble_get_localname 11
2.2. Classic BT APIs 11
2.2.1. ql_classic_bt_set_scanmode 11
2.2.2. ql_classic_bt_get_scanmode 12
2.2.3. ql_classic_bt_get_localaddr 12
2.2.4. ql_ble_db_service_add 13
2.2.5. ql_classic_bt_start_inqury* 13
2.2.6. ql_classic_bt_cancel_inqury* 13
2.2.7. ql_classic_bt_start_bond* 14
2.2.8. ql_classic_bt_cancel_bond* 14
2.2.9. ql_classic_bt_get_bond_info* 15
2.2.10. ql_classic_bt_remove_typical_bond_device* 15
2.2.11. ql_classic_bt_remove_all_bond_device* 16
2.2.12. ql_classic_bt_connect* 16
2.2.13. ql_classic_bt_disconnect* 17
2.2.14. ql_classic_bt_get_connection* 17
2.2.15. ql_classic_bt_spp_send* 17
2.3. BLE APIs 18
2.3.1. ql_ble_get_version 18
2.3.2. ql_ble_get_public_addr 18
2.3.3. ql_ble_get_random_addr 19
2.3.4. ql_ble_add_public_whitelist 19
2.3.5. ql_ble_add_random_whitelist 20
2.3.6. ql_ble_get_whitelist_info 20
2.3.7. ql_ble_remove_whitelist 21
2.3.8. ql_ble_clean_whitelist 21
2.3.9. ql_ble_conncet_public_addr 22
2.3.10. ql_ble_conncet_random_addr 22
2.3.11. ql_ble_cancel_connect 23
2.3.12. ql_ble_disconnect 23

2.3.13. ql_ble_update_conn_param 23
2.3.14. ql_ble_exchage_mtu 24
2.4. BLE GATT Server APIs 24
2.4.1. ql_ble_gatt_server_init 24
2.4.2. ql_ble_gatt_server_release 25
2.4.3. ql_bleadv_set_typical_addr_param 25
2.4.4. ql_bleadv_set_param 26
2.4.5. ql_bleadv_set_data 26
2.4.6. ql_bleadv_set_scan_rsp_data 27
2.4.7. ql_ble_gatt_add_service 27
2.4.8. ql_ble_gatt_add_chara 28
2.4.9. ql_ble_gatt_add_chara_value 28
2.4.10. ql_ble_gatt_add_chara_desc 29
2.4.11. ql_ble_gatt_add_or_clear_service_complete 30
2.4.12. ql_bleadv_start 30
2.4.13. ql_bleadv_stop 31
2.4.14. ql_ble_send_notification_data 31
2.4.15. ql_ble_send_indication_data 32
2.4.16. ql_ble_set_ibeacon_data 33
2.4.17. ql_ble_write_ibeacon_cfg 33
2.4.18. ql_ble_read_ibeacon_cfg 34
2.5. BLE GATT Client APIs 34
2.5.1. ql_ble_gatt_client_init 34
2.5.2. ql_ble_gatt_client_release 35
2.5.3. ql_blescan_set_param 35
2.5.4. ql_blescan_start 35
2.5.5. ql_blescan_stop 36
2.5.6. ql_ble_gatt_discover_all_service 36
2.5.7. ql_ble_gatt_discover_by_uuid 37
2.5.8. ql_ble_gatt_discover_all_includes 37
2.5.9. ql_ble_gatt_discover_all_characteristic 38
2.5.10. ql_ble_gatt_discover_chara_desc 38
2.5.11. ql_ble_gatt_read_chara_value_by_uuid 39
2.5.12. ql_ble_gatt_read_chara_value_by_handle 40
2.5.13. ql_ble_gatt_read_mul_chara_value 40
2.5.14. ql_ble_gatt_read_chara_desc 41
2.5.15. ql_ble_gatt_write_chara_desc 41
2.5.16. ql_ble_gatt_write_chara_value 42
2.5.17. ql_ble_gatt_write_chara_value_no_rsp 43
2.6. BT HFP APIs 44
2.6.1. ql_bt_hfp_init 44
2.6.2. ql_bt_hfp_release 44
2.6.3. ql_bt_hfp_connect 44
2.6.4. ql_bt_hfp_disconnect 45

2.6.5. ql_bt_hfp_set_volume 45
2.6.6. ql_bt_hfp_reject_call 46
2.6.7. ql_bt_hfp_answer_accept_call 46
2.6.8. ql_bt_hfp_answer_reject_call 47
2.6.9. ql_bt_hfp_dial* 47
2.6.10. ql_bt_hfp_redial* 48
2.6.11. ql_bt_hfp_vr_enable* 48
2.6.12. ql_bt_hfp_vr_disable* 49
2.6.13. ql_bt_hfp_ctrl_three_way_call* 49
2.7. BT A2DP AVRCP APIs 50
2.7.1. ql_bt_a2dp_avrcp_init 50
2.7.2. ql_bt_a2dp_avrcp_release 50
2.7.3. ql_bt_a2dp_disconnect 50
2.7.4. ql_bt_a2dp_connection_state_get 51
2.7.5. ql_bt_a2dp_get_addr 51
2.7.6. ql_bt_avrcp_start 52
2.7.7. ql_bt_avrcp_pause 52
2.7.8. ql_bt_avrcp_previ 53
2.7.9. ql_bt_avrcp_next 53
2.7.10. ql_bt_avrcp_vol_set 53
2.7.11. ql_bt_avrcp_vol_get 54
2.7.12. ql_bt_avrcp_play_state_get 54
2.7.13. ql_bt_avrcp_connection_state_get 55

3 Demo of BT/BLE 56
3.1. Demo of BLE GATT Client 56
3.2. Demo of BLE GATT Server 58
3.3. Demo of BT HFP 62
3.4. Demo of BT A2DP AVRCP 63

4 Appendix A References 66

Table Index

Table 1: Related Documents 66
Table 2: Terms and Abbreviations 66

1 Introduction

Quectel EC200U-CN modules support QuecOpen® solution. QuecOpen is an open-source embedded development platform based on Linux system. It is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see ***Document [1]***.

This document introduces the Bluetooth of EC200U-CN QuecOpen, including the classic BT SPP* process, the BLE GATT service and GATT client process, the classic Bluetooth HFP process, the classic Bluetooth A2DP and ACRCP process.

NOTE

"*" means under development.

2 Bluetooth APIs

2.1. General Bluetooth APIs

2.1.1. ql_bt_start

This function starts the Bluetooth service.

- **Prototype**

```
ql_errcode_bt_e ql_bt_start()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.1.2. ql_bt_stop

This function stops the Bluetooth service.

- **Prototype**

```
ql_errcode_bt_e ql_bt_stop()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.1.3. ql_bt_get_state

This function gets the Bluetooth state.

- **Prototype**

```
ql_errcode_bt_e ql_bt_get_state(ql_bt_state_e *bt_state)
```

- **Parameters**

bt_state:
[Out] Type of *ql_bt_state_e*.
QL_BT_STOPED indicates the Bluetooth service is stopped.
QL_BT_STARTED indicates the Bluetooth service is started.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.1.4. ql_bt_ble_set_localname

This function sets the Bluetooth/BLE name to NVM. The maximum length of the name is 22 bytes.

- **Prototype**

```
ql_errcode_bt_e ql_bt_ble_set_localname(ql_bt_ble_local_name_s local_name)
```

- **Parameters**

local_name:
[In] Type of *ql_bt_ble_local_name_s*. Bluetooth/BLE name.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.1.5. ql_bt_ble_get_localname

This function gets the Bluetooth/BLE name from NVM.

- **Prototype**

```
ql_errcode_bt_e ql_bt_ble_get_localname(ql_bt_ble_local_name_s *local_name)
```

- **Parameters**

local_name:
[Out] Type of *ql_bt_ble_local_name_s*. Bluetooth/BLE name.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCES	Indicates the operation is successful
Other value	Indicates an error occurs

2.2. Classic Bluetooth APIs

2.2.1. ql_classic_bt_set_scanmode

This function sets the scan mode of Bluetooth.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_set_scanmode(ql_bt_visible_mode_e scanmode_info)
```

- **Parameters**

scanmode_info:
[In] Type of *ql_bt_visible_mode_e*.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.2. ql_classic_bt_get_scanmode

This function gets the scan mode of the classic Bluetooth.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_get_scanmode(ql_bt_visible_mode_e *scanmode_info)
```

- **Parameters**

scanmode_info:
[Out] Type of *ql_bt_visible_mode_e*

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.2.3. ql_classic_bt_get_localaddr

This function obtains the address of the classic Bluetooth.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_get_localaddr(ql_bt_addr_s *local_addr)
```

- **Parameters**

local_addr:
[Out] Type of *ql_bt_addr_s.* The address of the classic Bluetooth.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.2.4. ql_ble_db_service_add

This function sets the address of the classic Bluetooth.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_set_localaddr(ql_bt_addr_s local_addr)
```

- **Parameters**

local_addr:
[In] Type of *ql_bt_addr_s.* The address of the classic Bluetooth.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.5. ql_classic_bt_start_inqury*

The classic Bluetooth starts to scan by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_start_inqury(ql_bt_inqury_type_e inqury_type)
```

- **Parameters**

inqury_type:
[In] Type of *ql_bt_inqury_type_e.* The scan type.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.6. ql_classic_bt_cancel_inqury*

The classic Bluetooth cancels the current scan by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_cancel_inqury()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.7. ql_classic_bt_start_bond*

The classic Bluetooth bonds with a Bluetooth device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_start_bond(ql_bt_addr_s bond_addr)
```

- **Parameters**

bond_addr:
[In] Type of *ql_bt_addr_s.* The address of the Bluetooth device.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.8. ql_classic_bt_cancel_bond*

The classic Bluetooth cancels the bonds with a Bluetooth device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_cancel_bond(ql_bt_addr_s bond_addr)
```

- **Parameters**

bond_addr:
[In] Type of *ql_bt_addr_s.* The address of the Bluetooth device.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.9. ql_classic_bt_get_bond_info*

The classic Bluetooth obtains the paired Bluetooth information by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_get_bond_info(unsigned int number,unsigned int *device_number,ql_bt_info_s info[])
```

- **Parameters**

number:
[In] The target number of paired devices.

device_number:
[Out] The actual number of paired devices returned.

info:
[Out] Type of *ql_bt_info_s*. The saved information of paired Bluetooth.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.2.10. ql_classic_bt_remove_typical_bond_device*

The classic Bluetooth removes a specific paired device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_remove_typical_bond_device(ql_bt_addr_s bond_addr)
```

- **Parameters**

bond_addr:
[In] Type of *ql_bt_addr_s.* The address of the paired Bluetooth device.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.2.11. ql_classic_bt_remove_all_bond_device*

The classic Bluetooth removes all the paired device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_remove_all_bond_device()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.2.12. ql_classic_bt_connect*

The classic Bluetooth connects to other Bluetooth device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_connect(ql_bt_connect_info_s connect_info)
```

- **Parameters**

connect_info:
[In] Type of *ql_bt_connect_info_s*. The type and address of the connected Bluetooth device.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.13. ql_classic_bt_disconnect*

The classic Bluetooth disconnects from other Bluetooth device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_disconnect()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
QL_BT_PENDING	Indicates the operation is pending
Other value	Indicates an error occurs

2.2.14. ql_classic_bt_get_connection*

The classic Bluetooth gets the connection state based on the address of the Bluetooth device by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_get_connection(ql_bt_addr_s connect_addr)
```

- **Parameters**

connect_addr:
[In] Bluetooth address in structure of *ql_bt_addr_s*

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates it is connected successfully
QL_BT_DISCONNECT	Indicates it is disconnected
Other value	Indicates an error occurs

2.2.15. ql_classic_bt_spp_send*

The classic Bluetooth sends data by calling this function.

- **Prototype**

```
ql_errcode_bt_e ql_classic_bt_spp_send(ql_bt_spp_info_s spp_info)
```

- **Parameters**

spp_info:
[In] Type of *ql_bt_spp_info_s*. The data to be sent.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3. BLE APIs

2.3.1. ql_ble_get_version

This function gets the BLE version number.

- **Prototype**

```
ql_errcode_bt_e ql_ble_get_version(char *version, unsigned int get_version_len)
```

- **Parameters**

version:
[Out] BLE version number.

get_version_len:
[In] The length of the version number to be read.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.2. ql_ble_get_public_addr

This function gets the public address.

- **Prototype**

```
ql_errcode_bt_e ql_ble_get_public_addr(ql_bt_addr_s * public_addr)
```

- **Parameters**

public_addr:
[Out] Type of *ql_bt_addr_s*. The public address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.3. ql_ble_get_random_addr

This function gets the random address.

- **Prototype**

```
ql_errcode_bt_e ql_ble_get_random_addr(ql_bt_addr_s *random_addr)
```

- **Parameters**

random_addr:
[Out] Type of *ql_bt_addr_s*. The random address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.4. ql_ble_add_public_whitelist

This function adds public address to the whitelist.

- **Prototype**

```
ql_errcode_bt_e ql_ble_add_public_whitelist(ql_bt_addr_s public_addr)
```

- **Parameters**

public_addr:
[In] Type of *ql_bt_addr_s*. The public address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.5. ql_ble_add_random_whitelist

This function adds the random address to the whitelist.

- **Prototype**

```
ql_errcode_bt_e ql_ble_add_random_whitelist(ql_bt_addr_s random_addr)
```

- **Parameters**

random_addr:
[In] Type of *ql_bt_addr_s*. The random address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.6. ql_ble_get_whitelist_info

This function gets the whitelist information.

- **Prototype**

```
ql_errcode_bt_e ql_ble_get_whitelist_info(unsigned char whitelist_count, unsigned char *real_whitelist_count, ql_ble_whitelist_info_s whitelist[])
```

- **Parameters**

whitelist_count:
[In] The number of whitelists to be obtained.

real_whitelist_count:
[In] The actual number of whitelists obtained.

whitelist:
[In] Type of *ql_ble_whitelist_info_s*. The whitelist information.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.7. ql_ble_remove_whitelist

This function removes the whitelist of the specific address.

- **Prototype**

```
ql_errcode_bt_e ql_ble_remove_whitelist( ql_ble_whitelist_info_s whitelist )
```

- **Parameters**

whitelist:
[In] Type of *ql_ble_whitelist_info_s*. The whitelist.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.8. ql_ble_clean_whitelist

This function cleans all whitelists.

- **Prototype**

```
ql_errcode_bt_e ql_ble_clean_whitelist()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.9. ql_ble_conncet_public_addr

This function connects to the public address. This function needs to process the connection result asynchronously.

- **Prototype**

```
ql_errcode_bt_e ql_ble_conncet_public_addr(ql_bt_addr_s public_addr)
```

- **Parameters**

public_addr:
[In] Type of *ql_bt_addr_s*. The public address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.10. ql_ble_conncet_random_addr

This function connects to the random address. This function needs to process the connection result asynchronously.

- **Prototype**

```
ql_errcode_bt_e ql_ble_conncet_random_addr(ql_bt_addr_s random_addr)
```

- **Parameters**

public_addr:
[In] Type of *ql_bt_addr_s*. The random address of the Bluetooth.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.11. ql_ble_cancel_connect

This function cancels the BLE connection. This function needs to process the disconnection result asynchronously.

- **Prototype**

```
ql_errcode_bt_e ql_ble_cancel_connect(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] Type of *ql_bt_addr_s*. The address of the Bluetooth.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.12. ql_ble_disconnect

This function disconnects from the BLE. This function needs to process the disconnection result asynchronously.

- **Prototype**

```
ql_errcode_bt_e ql_ble_disconnect(unsigned short conn_id)
```

- **Parameters**

conn_id:
[In] The connected ID obtained when the connection is established

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.13. ql_ble_update_conn_param

This function updates the connection parameters.

- **Prototype**

```
ql_errcode_bt_e ql_ble_update_conn_param(ql_ble_update_conn_infos_s conn_param)
```

- **Parameters**

conn_param:
[In] Type of *ql_ble_update_conn_infos_s*. The connection parameters.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.3.14. ql_ble_exchage_mtu

This function request to update MTU.

- **Prototype**

```
ql_errcode_bt_e ql_ble_exchage_mtu(unsigned short conn_id, unsigned short mtu)
```

- **Parameters**

conn_id:
[In] The connected ID obtained when the connection is established.

mtu:
[In] MTU value. Range: 0–247.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4. BLE GATT Server APIs

2.4.1. ql_ble_gatt_server_init

This function initializes the BLE GATT server and register the BLE callback function. Only by returning *QL_BT_SUCCESS* can other APIs be called.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_server_init(ql_bt_callback bt_cb)
```

- **Parameters**

bt_cb:
[In] The type of *ql_bt_callback*. Callback function.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.2. ql_ble_gatt_server_release

This function releases the resources occupied by the BLE GATT server.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_server_release()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.3. ql_bleadv_set_typical_addr_param

This function sets the broadcast parameters of a typical address.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_set_typical_addr_param(ql_bleadv_typical_addr_param_s adv_param)
```

- **Parameters**

adv_param:
[In] The type of *ql_bleadv_typical_addr_param_s*. The broadcast parameter.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.4. ql_bleadv_set_param

This function sets the broadcast parameters.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_set_param(ql_bleadv_param_s adv_param)
```

- **Parameters**

adv_param:
[In] The type of *ql_bleadv_param_s*. The broadcast parameter.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.5. ql_bleadv_set_data

This function sets broadcast data.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_set_data(ql_bleadv_set_data_s adv_data)
```

- **Parameters**

adv_data:
[In] The type of *ql_bleadv_set_data_s*. The broadcast data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.6. ql_bleadv_set_scan_rsp_data

This function sets the broadcast data received from the scan.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_set_scan_rsp_data(ql_bleadv_set_data_s adv_data)
```

- **Parameters**

adv_data:
[In] Type of *ql_bleadv_set_data_s.* The broadcast data received from the scan.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.7. ql_ble_gatt_add_service

This function adds the GATT service.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_add_service(unsigned short server_id, ql_ble_gatt_uuid_s uuid, unsigned char primary)
```

- **Parameters**

server_id:
[In] Sever ID to mark the same set of servers.

uuid:
[In] Type of *ql_ble_gatt_uuid_s.* UUID data.

primary:
[In] Whether it is the primary server.
0 Not the primary server
1 Primary server

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.8. ql_ble_gatt_add_chara

This function adds GATT characteristics.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_add_chara(unsigned short server_id, unsigned short chara_id, unsigned char prop, ql_ble_gatt_uuid_s uuid)
```

- **Parameters**

server_id:
[In] Sever ID to mark the same set of servers. It is consistent with *server_id* in *ql_ble_gatt_add_service*.

chara_id:
[In] Characteristics ID to mark the same feature.

prop:
[In] The property of the characteristics.

uuid:
[In] Type of *ql_ble_gatt_uuid_s.* UUID data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.9. ql_ble_gatt_add_chara_value

This function adds characteristic value of GATT.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_add_chara_value(unsigned short server_id, unsigned short chara_id, unsigned char permission, ql_ble_gatt_uuid_s uuid, unsigned short value_len, unsigned char *value)
```

- **Parameters**

server_id:
[In] Sever ID to mark the same set of servers. It is consistent with *server_id* in *ql_ble_gatt_add_service*.

chara_id:
[In] Characteristics ID to mark the same set of characteristics. It is consistent with *chara_id* in *ql_ble_gatt_add_chara*.

uuid:
[In] Type of *ql_ble_gatt_uuid_s.* UUID data.

length:
[In] Length of the characteristic value.

value:
[In] Characteristics value.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.10. ql_ble_gatt_add_chara_desc

This function adds the characteristic descriptor of GATT.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_add_chara_desc(unsigned short server_id, unsigned short chara_id, unsigned char permission, ql_ble_gatt_uuid_s uuid, unsigned short value_len, unsigned char *value)
```

- **Parameters**

server_id:
[In] Sever ID to mark the same set of servers. It is consistent with *server_id* in *ql_ble_gatt_add_service*.

chara_id:
[In] Characteristics ID to mark the same set of characteristics. It is consistent with *chara_id* in *ql_ble_gatt_add_chara*.

uuid:
[In] Type of *ql_ble_gatt_uuid_s.* UUID data.

length:
[In] Length of the characteristic value.

value:
[In] Content of the characteristic descriptor.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.11. ql_ble_gatt_add_or_clear_service_complete

This function finishes adding the service, or to clear the service that has been added to the memory.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_add_or_clear_service_complete(unsigned short type, ,
ql_ble_sys_service_mode_e mode)
```

- **Parameters**

type:
[In] Type of execution.

0 Clear the services, features, etc. that have been added to the memory
1 Add the service in the memory to the protocol stack

mode:
[In] Whether to keep the system default service mode when adding a service operation.

0 Clear system default GATT and GAP servers
1 Keep system default GATT and GAP servers

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.12. ql_bleadv_start

This function starts the broadcast.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_start()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.13. ql_bleadv_stop

This function stops the broadcast.

- **Prototype**

```
ql_errcode_bt_e ql_bleadv_stop()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.14. ql_ble_send_notification_data

This function sends the notification data.

- **Prototype**

```
ql_errcode_bt_e ql_ble_send_notification_data(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value)
```

- **Parameters**

conn_id:
[In] The connection ID.

att_handle:
[In] GATT handler

length:
[In] The length of the sent data. Do not exceed the length of MTU.

value:
[In] Content of the sent content.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.15. ql_ble_send_indication_data

This function sends the indication data.

- **Prototype**

```
ql_errcode_bt_e ql_ble_send_indication_data(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value)
```

- **Parameters**

conn_id:
[In] The connection ID.

att_handle:
[In] GATT handler.

length:
[In] The length of the sent data. Do not exceed the length of MTU.

value:
[In] Content of the sent content.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.16. ql_ble_set_ibeacon_data

This function sets ibeacon data.

- **Prototype**

```
ql_errcode_bt_e ql_ble_set_ibeacon_data(unsigned char uuid_l[QL_BLE_LONG_UUID_SIZE], unsigned short major, unsigned short minor)
```

- **Parameters**

uuid_l:
[In] 16-byte UUID.

major:
[In] Major part.

minor:
[In] Minor part.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.17. ql_ble_write_ibeacon_cfg

This function writes ibeacon data to NVM.

- **Prototype**

```
ql_errcode_bt_e ql_ble_write_ibeacon_cfg(ql_ble_ibeacon_cfg_s cfg)
```

- **Parameters**

cfg:
[In] Type of *ql_ble_ibeacon_cfg_s*. The ibeacon data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.4.18. ql_ble_read_ibeacon_cfg

This function reads the ibeacon data.

- **Prototype**

```
ql_errcode_bt_e ql_ble_read_ibeacon_cfg(ql_ble_ibeacon_cfg_s *cfg)
```

- **Parameters**

cfg:
[Out] Type of *ql_ble_ibeacon_cfg_s*. The ibeacon data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5. BLE GATT Client APIs

2.5.1. ql_ble_gatt_client_init

This function initializes the BLE GATT client and register the callback function. Only by returning *QL_BT_SUCCESS* can other APIs be called.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_client_init(ql_bt_callback bt_cb)
```

- **Parameters**

bt_cb:
[In] The type of *ql_bt_callback*. Callback function.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.2. ql_ble_gatt_client_release

This function releases the resources occupied by the BLE GATT server.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_client_release()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.3. ql_blescan_set_param

This function sets the scan parameters.

- **Prototype**

```
ql_errcode_bt_e ql_blescan_set_param(ql_blescan_scan_s scan_param)
```

- **Parameters**

scan_param:
[In] The type of *ql_blescan_scan_s*. The scan parameters.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.4. ql_blescan_start

This function starts to scan.

- **Prototype**

```
ql_errcode_bt_e ql_blescan_start()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.5. ql_blescan_stop

This function stops scanning.

- **Prototype**

```
ql_errcode_bt_e ql_blescan_stop()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.6. ql_ble_gatt_discover_all_service

This function discovers all services. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_discover_all_service(unsigned short conn_id)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.7. ql_ble_gatt_discover_by_uuid

This function discovers specific UUID service. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_discover_by_uuid(unsigned short conn_id, ql_ble_gatt_uuid_s uuid)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

uuid:
[In] Type of *ql_ble_gatt_uuid_s.* UUID data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.8. ql_ble_gatt_discover_all_includes

This function discovers all GATT client information. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_discover_all_includes(unsigned short conn_id, unsigned short start_handle, unsigned short end_handle)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

start_handle:
[In] The started handler.

end_handle:
[In] The end handler.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.9. ql_ble_gatt_discover_all_characteristic

This function discovers all characteristics. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_discover_all_characteristic(unsigned short conn_id, unsigned short start_handle, unsigned short end_handle)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

start_handle:
[In] The started handler.

end_handle:
[In] The end handler.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.10. ql_ble_gatt_discover_chara_desc

This function discovers characteristic descriptor of GATT. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_discover_chara_desc(unsigned short conn_id, unsigned short start_handle, unsigned short end_handle)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

start_handle:
[In] The started handler.

end_handle:
[In] The end handler.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.11. ql_ble_gatt_read_chara_value_by_uuid

This function reads the characteristic value with UUID. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_read_chara_value_by_uuid(unsigned short conn_id,  ql_ble_gatt_uuid_s uuid, unsigned short start_handle, unsigned short end_handle)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

uuid:
[In] Type of *ql_ble_gatt_uuid_s*. UUID data.

start_handle:
[In] The started handler.

end_handle:
[In] The end handler.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.12. ql_ble_gatt_read_chara_value_by_handle

This function reads the characteristic value with handles. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_read_chara_value_by_handle(unsigned short conn_id, unsigned short handle, unsigned short offset, unsigned char islong)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

handle:
[In] GATT handler.

offset:
[In] Offset address when read multiple times.

islong:
[In] Multiple read flag.

0	No need to read multiple times
1	Need to read multiple times

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.13. ql_ble_gatt_read_mul_chara_value

This function reads multiple characteristic values. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_read_mul_chara_value(unsigned short conn_id, unsigned char *handle, unsigned char length)
```

- **Parameters**

conn_id:
[In] The ID when the connection is established.

handle:
[In] GATT handler.

length:
[In] Length of the handler.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.14. ql_ble_gatt_read_chara_desc

This function reads the characteristic descriptors. This function needs to process the data in the callback.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_read_chara_desc(unsigned short conn_id, unsigned short handle, unsigned char islong)
```

- **Parameters**

conn_id:
[In] The ID returned when the connection is established.

handle:
[In] GATT handler.

islong:
[In] Multiple read flag.

0 No need to read multiple times
1 Need to read multiple times

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.15. ql_ble_gatt_write_chara_desc

This function writes the characteristic descriptor.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_write_chara_desc(unsigned short conn_id, unsigned short handle, unsigned short length, unsigned char *value)
```

- **Parameters**

conn_id:
[In] The ID returned when the connection is established.

handle:
[In] GATT handler.

length:
[In] Length of the descriptor data.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.16. ql_ble_gatt_write_chara_value

This function writes the characteristic value.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_write_chara_value(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value, unsigned short offset, unsigned char islong)
```

- **Parameters**

conn_id:
[In] The ID returned when the connection is established.

att_handle:
[In] GATT handler.

length:
[In] Length of the written characteristic data.

value:
[In] the data content written to the characteristic value.

islong:
[In] Whether it is a long characteristic value
1 long characteristic value

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.5.17. ql_ble_gatt_write_chara_value_no_rsp

This function writes characteristic value without the confirmation from the Bluetooth link layer.

- **Prototype**

```
ql_errcode_bt_e ql_ble_gatt_write_chara_value_no_rsp(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value)
```

- **Parameters**

conn_id:
[In] The ID returned when the connection is established.

att_handle:
[In] GATT handler.

length:
[In] Length of the written characteristic data.

value:
[In] the data content written to the characteristic value.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6. Bluetooth HFP APIs

2.6.1. ql_bt_hfp_init

This function initializes Bluetooth HFP and registers the callback function. Only by returning *QL_BT_SUCCESS* can other APIs be called.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_init(ql_bt_callback bt_cb)
```

- **Parameters**

bt_cb:
[In] Type of *ql_bt_callback*. The callback function.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.2. ql_bt_hfp_release

This function releases the resources occupied by Bluetooth HFP.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_release()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.3. ql_bt_hfp_connect

This function connects to AG and establishes connection with HFP.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_connect(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. Bluetooth address of AG.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.4. ql_bt_hfp_disconnect

This function disconnects from HFP.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_disconnect(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. Bluetooth address of AG.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.5. ql_bt_hfp_set_volume

This function sets call volume.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_set_volume(ql_bt_addr_s addr, unsigned char vol)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of AG that has established a connection.

vol:
[In] Call volume. Range: 1–15.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.6. ql_bt_hfp_reject_call

This function hangs up the call.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_reject_call(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.7. ql_bt_hfp_answer_accept_call

This function answers the incoming call.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_answer_accept_call(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.8. ql_bt_hfp_answer_reject_call

This function rejects the incoming call.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_answer_reject_call(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.9. ql_bt_hfp_dial*

This function dials a specific phone number.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_dial(ql_bt_addr_s addr,unsigned char *phone_number)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

phone_number:
[In] The phone number.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.10. ql_bt_hfp_redial*

This function redials a specific phone number.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_redial(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.11. ql_bt_hfp_vr_enable*

This function enables voice assistant service.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_vr_enable(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.12. ql_bt_hfp_vr_disable*

This function disables voice assistant service.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_vr_disable(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.6.13. ql_bt_hfp_ctrl_three_way_call*

This function controls three-way calling.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_ctrl_three_way_call(ql_bt_addr_s addr, unsigned char cmd)
```

- **Parameters**

addr:
[In] The type of *ql_bt_addr_s*. The Bluetooth address of the AG that has established a connection.

cmd:
[In] The control command for three-way calls.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7. Bluetooth A2DP AVRCP APIs

2.7.1. ql_bt_a2dp_avrcp_init

This function initializes Bluetooth A2DP and AVRCP, and registers the callback function. Only by returning *QL_BT_SUCCESS* can other APIs be called.

- **Prototype**

```
ql_errcode_bt_e ql_bt_a2dp_avrcp_init(ql_bt_callback bt_cb)
```

- **Parameters**

bt_cb:
[In] Type of *ql_bt_callback*. The callback function.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.2. ql_bt_a2dp_avrcp_release

This function releases the resources occupied by Bluetooth A2DP and AVRCP.

- **Prototype**

```
ql_errcode_bt_e ql_bt_hfp_release()
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.3. ql_bt_a2dp_disconnect

This function disconnects from A2DP.

- **Prototype**

```
ql_errcode_bt_e ql_bt_a2dp_disconnect(ql_bt_addr_s addr)
```

- **Parameters**

addr:
[In] Type of *ql_bt_addr_s*. The Bluetooth address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.4. ql_bt_a2dp_connection_state_get

This function gets A2DP connection state.

- **Prototype**

```
quec_bta2dp_connection_state_e ql_bt_a2dp_connection_state_get(void)
```

- **Parameters**

NA

- **Return Value**

Type of *quec_bta2dp_connection_state_e*.

0	Disconnected
1	Connecting
2	Connected
3	Disconnecting

2.7.5. ql_bt_a2dp_get_addr

This function gets the Bluetooth address of the host.

- **Prototype**

```
ql_errcode_bt_e ql_bt_a2dp_get_addr(ql_bt_addr_s *addr)
```

- **Parameters**

addr:
[Out] Type of *ql_bt_addr_s*. The host address.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.6. ql_bt_avrcp_start

This function controls the host to start playing audio files through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_start(void)
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.7. ql_bt_avrcp_pause

This function controls the host to pause playing audio files through the AVRCP protocol

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_pause(void)
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.8. ql_bt_avrcp_previ

This function controls the host to play the previous audio file through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_previ(void)
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.9. ql_bt_avrcp_next

This function controls the host to play the next audio file through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_next(void)
```

- **Parameters**

NA

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.10. ql_bt_avrcp_vol_set

This function controls the volume of the host through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_vol_set(uint8 vol)
```

- **Parameters**

vol:
[In] The audio playback volume. Range: 0–127.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.11. ql_bt_avrcp_vol_get

This function gets the volume of the host through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_vol_get(uint8_t* vol)
```

- **Parameters**

vol:
[Out] The audio playback volume. Range: 0–127.

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.12. ql_bt_avrcp_play_state_get

This function obtains the audio playback state of the host through the AVRCP protocol.

- **Prototype**

```
ql_errcode_bt_e ql_bt_avrcp_play_state_get(QUEC_AVRCP_PLAYBACK_STATUS_e *ret)
```

- **Parameters**

ret:
[Out] Type of QUEC_AVRCP_PLAYBACK_STATUS_e. Audio playback state of the host.

0	Not playing
1	Playing
2	Pause playing
3	Switching to the next audio file
4	Switching to the previous audio file
255	An error occurred while obtaining

- **Return Value**

Type of *ql_errcode_bt_e*.

QL_BT_SUCCESS	Indicates the operation is successful
Other value	Indicates an error occurs

2.7.13. ql_bt_avrcp_connection_state_get

This function gets the host connection state through the AVRCP protocol.

- **Prototype**

```
quec_btavrcp_connection_state_t ql_bt_avrcp_connection_state_get(void)
```

- **Parameters**

NA

- **Return Value**

Type of *quec_btavrcp_connection_state_t*.

0	Disconnected
1	Connecting
2	Connected
3	Disconnecting

3 Demo of Bluetooth/BLE

3.1. Demo of BLE GATT Client

Step 1: Call the following function to register the callback function.

```
ql_errcode_bt_e  ql_ble_gatt_client_init(ql_bt_callback  bt_cb)
```

Step 2: Call the following function to start the Bluetooth service.

```
ql_errcode_bt_e  ql_bt_start()
```

Step 3: Call the following function to set the scan parameters.

```
ql_errcode_bt_e ql_blescan_set_param(ql_blescan_scan_s scan_param)
```

Step 4: Call the following function to start the scan. Wait for other BLE devices to be scanned. If the scanned BLE device address is a public address, skip to ***Step 5***, and if the scanned address is a random address, skip to ***Step 6***.

```
ql_errcode_bt_e ql_blescan_start()
```

Step 5: Call the following function to connect to the public address.

```
ql_errcode_bt_e ql_ble_conncet_public_addr(ql_bt_addr_s public_addr)
```

Step 6: Call the following function to connect to the random address.

```
ql_errcode_bt_e  ql_ble_conncet_public_addr(ql_bt_addr_s  public_addr)
```

Step 7: Call the following function to discover all services.

```
ql_errcode_bt_e  ql_ble_gatt_discover_all_service(unsigned  short  conn_id)
```

Step 8: Call the following function to discover all characteristics.

```
ql_errcode_bt_e ql_ble_gatt_discover_all_characteristic(unsigned short conn_id, unsigned short start_handle, unsigned short end_handle)
```

Step 9: Call the following function to discover characteristic descriptor of GATT.

```
ql_errcode_bt_e ql_ble_gatt_discover_chara_desc(unsigned short conn_id, unsigned short start_handle, unsigned short end_handle)
```

Step 10: Call the following function to write the characteristic value.

```
ql_errcode_bt_e ql_ble_gatt_write_chara_value(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value, unsigned short offset, unsigned char islong)
```

Step 11: Call the following function to write the characteristic descriptor.

```
ql_errcode_bt_e ql_ble_gatt_write_chara_desc(unsigned short conn_id, unsigned short handle, unsigned short length, unsigned char *value)
```

Step 12: Call the following function to read the characteristic value.

```
ql_errcode_bt_e ql_ble_gatt_read_chara_value_by_handle(unsigned short conn_id, unsigned short handle, unsigned short offset, unsigned char islong)
```

Step 13: Call the following function to read the characteristic descriptor.

```
ql_errcode_bt_e ql_ble_gatt_read_chara_desc(unsigned short conn_id, unsigned short handle, unsigned char islong)
```

Step 14: Call the following function to disconnect from the BLE.

```
ql_errcode_bt_e ql_ble_disconnect(ql_bt_addr_s addr)
```

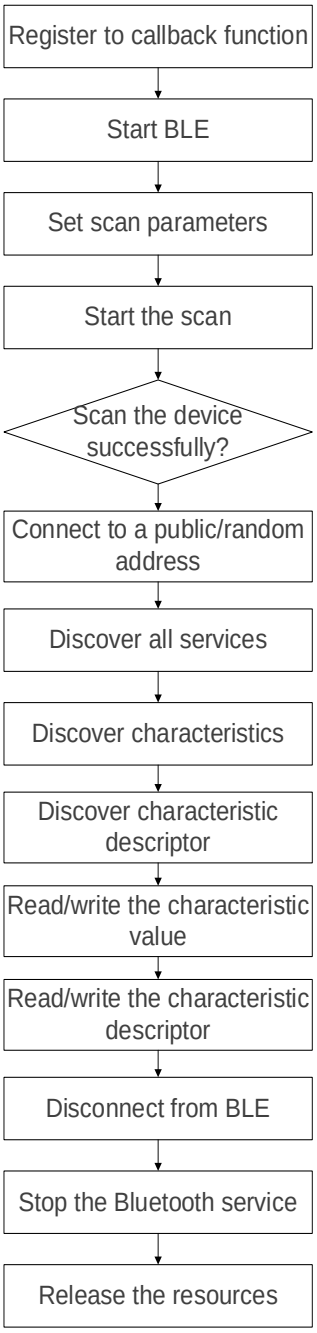
Step 15: Call the following function to stop the Bluetooth service.

```
ql_errcode_bt_e ql_bt_stop()
```

Step 16: Call the following function to release the resources.

```
ql_errcode_bt_e ql_ble_gatt_client_release()
```

The process is as follows:



3.2. Demo of BLE GATT Server

Step 1: Call the following function to register the callback function.

```
ql_errcode_bt_e  ql_ble_gatt_server_init(ql_bt_callback  bt_cb)
```

Step 2: Call the following function to start the Bluetooth service.

```
ql_errcode_bt_e ql_bt_start()
```

Step 3: Call the following function to set the broadcast parameters.

```
ql_errcode_bt_e ql_bleadv_set_param(ql_bleadv_param_s adv_param)
```

Step 4: Call the following function to set broadcast data.

```
ql_errcode_bt_e ql_bleadv_set_data(ql_bleadv_set_data_s avd_data)
```

Step 5: Call the following function to set the broadcast data received from the scan.

```
ql_errcode_bt_e ql_bleadv_set_scan_rsp_data(ql_bleadv_set_data_s adv_data)
```

Step 6: Call the following function to add GATT service.

```
ql_errcode_bt_e ql_ble_gatt_add_service(unsigned short server_id, ql_ble_gatt_uuid_s uuid, unsigned char primary)
```

Step 7: Call the following function to add GATT characteristics.

```
ql_errcode_bt_e ql_ble_gatt_add_chara(unsigned short server_id, unsigned short chara_id, unsigned char prop, ql_ble_gatt_uuid_s uuid)
```

Step 8: Call the following function to add characteristic value of GATT.

```
ql_errcode_bt_e ql_ble_gatt_add_chara_value(unsigned short server_id, unsigned short chara_id, unsigned char permission, ql_ble_gatt_uuid_s uuid, unsigned short value_len, unsigned char *value)
```

Step 9: Call the following function to add the characteristic descriptor of GATT.

```
ql_errcode_bt_e ql_ble_gatt_add_chara_desc(unsigned short server_id, unsigned short chara_id, unsigned char permission, ql_ble_gatt_uuid_s uuid, unsigned short value_len, unsigned char *value)
```

Step 10: Call the following function to finish adding the service.

```
ql_errcode_bt_e ql_ble_gatt_add_or_clear_service_complete(unsigned short type)
```

Step 11: Call the following function to start the broadcast. Wait for other BLE devices to scan and connect. After connecting, enter the connection mode.

```
ql_errcode_bt_e ql_bleadv_start()
```

Step 12: Call the following function to send the notification data.

```
ql_errcode_bt_e ql_ble_send_notification_data(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value)
```

Step 13: Call the following function to send the indication data.

```
ql_errcode_bt_e ql_ble_send_indication_data(unsigned short conn_id, unsigned short att_handle, unsigned short length, unsigned char *value)
```

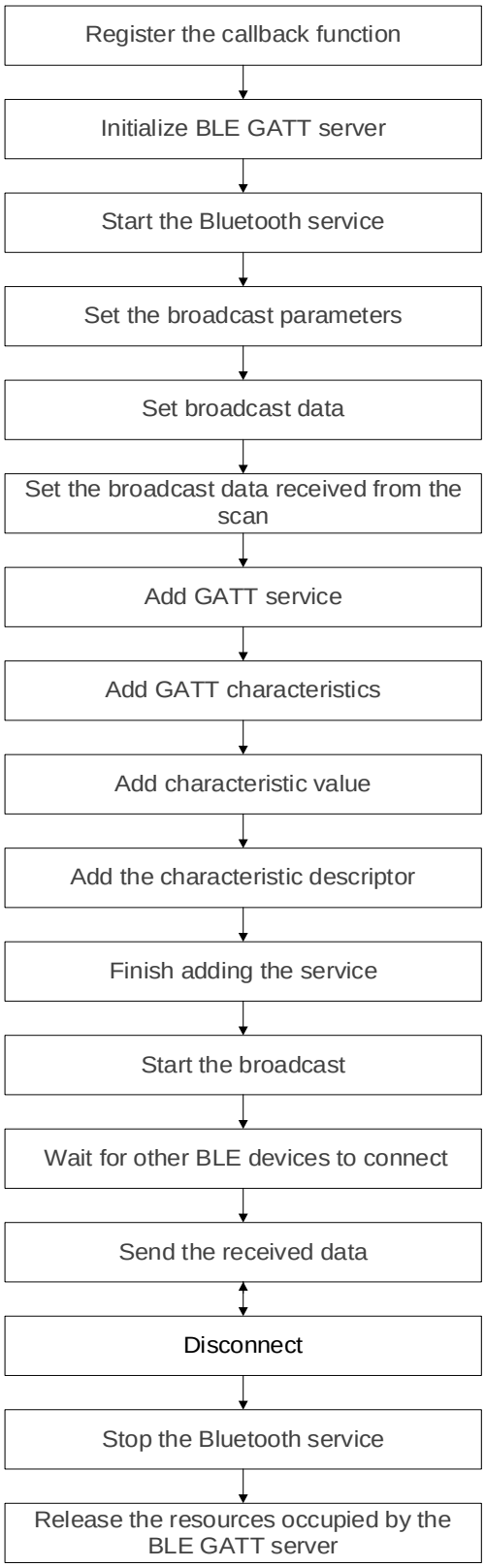
Step 14: Call the following function to stop the Bluetooth service.

```
ql_errcode_bt_e ql_bt_stop()
```

Step 15: Call the following function to release the resources occupied by the BLE GATT server.

```
ql_errcode_bt_e  ql_ble_gatt_server_release()
```

The process is as follows:



3.3. Demo of Bluetooth HFP

Step 1: Call the following function to register the callback function.

```
ql_errcode_bt_e  ql_bt_hfp_init(ql_bt_callback  bt_cb)
```

Step 2: Call the following function to start the Bluetooth service.

```
ql_errcode_bt_e  ql_bt_start()
```

Step 3: Call the following function to connect to AG. If you are waiting for the AG to actively connect to your device, this step can be omitted.

```
ql_errcode_bt_e  ql_bt_hfp_connect(ql_bt_addr_s  addr)
```

Step 4: Call the following function to answer the incoming call. When ringing, there is no need to connect the phone, and this step can be omitted.

```
ql_errcode_bt_e ql_bt_hfp_answer_accept_call(ql_bt_addr_s addr)
```

Step 5: Call the following function to set the call volume. If you do not need to adjust the volume, this step can be omitted.

```
ql_errcode_bt_e ql_bt_hfp_set_volume(ql_bt_addr_s addr, unsigned char vol)
```

Step 6: Call the following function to hang up the call.

```
ql_errcode_bt_e  ql_bt_hfp_reject_call(ql_bt_addr_s  addr)
```

Step 7: Call the following function to disconnect from HFP.

```
ql_errcode_bt_e  ql_bt_hfp_disconnect(ql_bt_addr_s  addr)
```

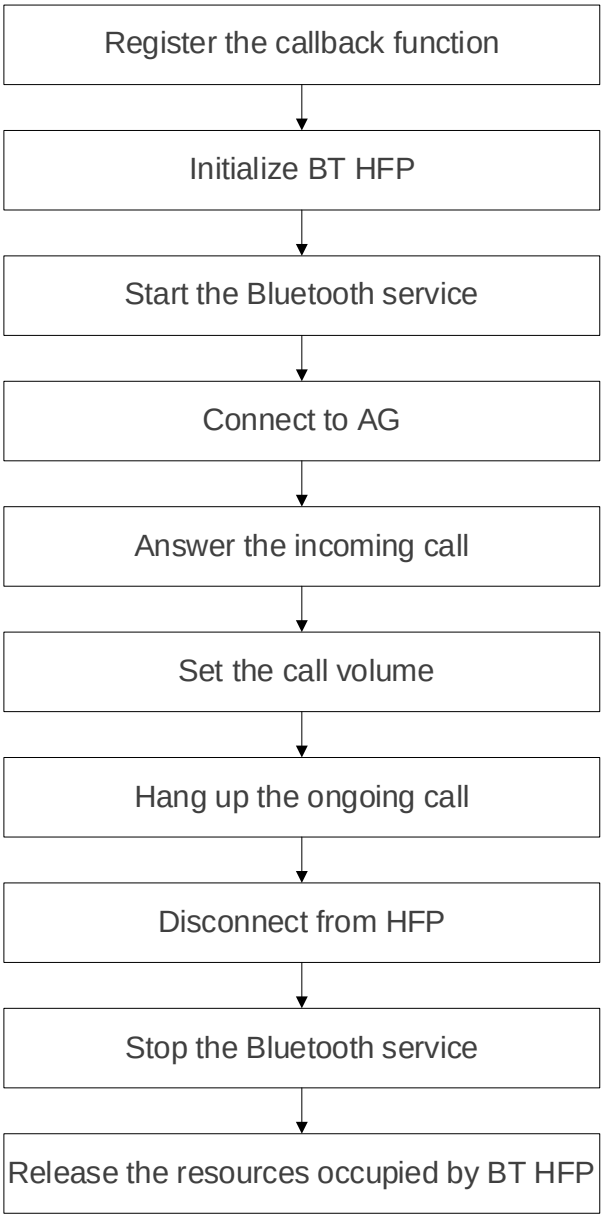
Step 8: Call the following function to stop the Bluetooth service.

```
ql_errcode_bt_e  ql_bt_stop()
```

Step 9: Call the following function to release the resources occupied by Bluetooth HFP.

```
ql_errcode_bt_e  ql_bt_hfp_release()
```

The process is as follows:



3.4. Demo of Bluetooth A2DP AVRCP

Step 1: Call the following function to register the callback function.

```
ql_errcode_bt_e  ql_bt_a2dp_avrcp_init(ql_bt_callback  bt_cb)
```

Step 2: Call the following function to start the Bluetooth service.

```
ql_errcode_bt_e  ql_bt_start()
```

Step 3: Call the following function to set the Bluetooth name.

```
ql_errcode_bt_e  ql_classic_bt_set_localname(ql_bt_local_name_s  local_name
```

Step 4: Call the following function to configure Bluetooth to be scannable and connectable. At this time, you can see from the log the prompt "checking connect".

```
ql_errcode_bt_e  ql_classic_bt_set_scanmode(ql_bt_visible_mode_e  scanmode_info)
```

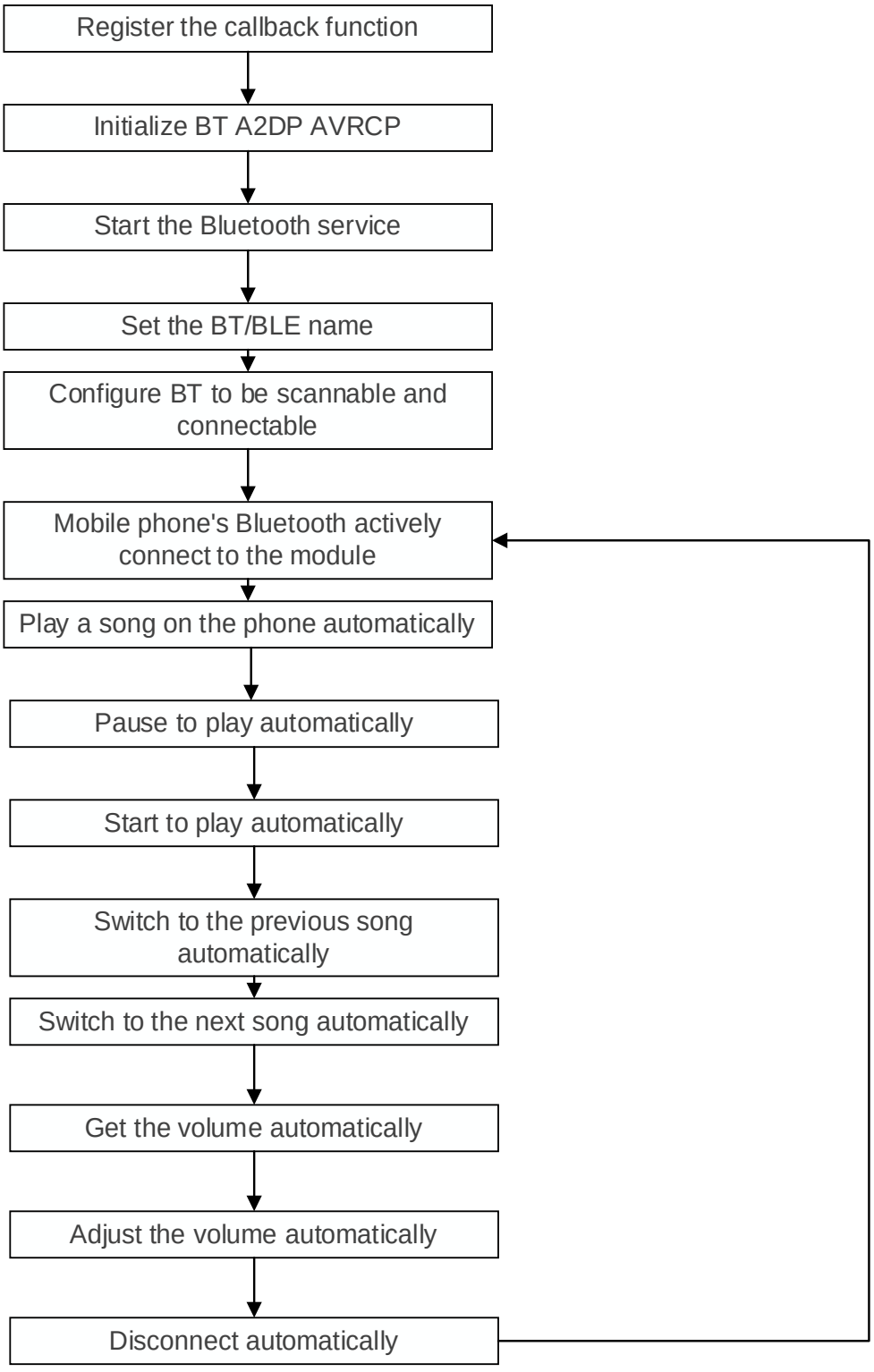
Step 5: Use the mobile phone's Bluetooth to connect to the module. After the connection is successful, you can see the prompt "checking paly" in the log.

Step 6: Open the music player software on your phone to play any song.

Step 7: After the playback is successful, the module will automatically operate in the following order: Pause > Start > Switch to the previous song > Switch to the next song > Get the volume > Adjust the volume up > Adjust the volume down > Disconnect

Step 8: If you need to re-test, just connect the module again with your mobile phone and repeat ***Step 6*** to play any song.

The process is as follows:



4 Appendix A References

Table 1: Related Documents

SN	Document Name	Description
[1]	Quectel_EC200U-CN_QuecOpen_Quick_Start_Guide	EC200U-CN QuecOpen quick start guide

Table 2: Terms and Abbreviations

Abbreviation	Description
A2DP	Advanced Audio Distribution Profile
AVRCP	Audio/Video Remote Control Profile
BLE	Bluetooth Low Energy
BT	Bluetooth
GATT	Generic Attribute Profile
ID	Identity
SPP	Serial Port Profile
UUID	Universally Unique Identifier
AG	Audio Gateway
HFP	Hand Free Profile