EC2x&EG25-G QuecOpen BT API Reference Manual

LTE Standard Module Series

Version: 1.0

Date: 2021-02-24

Status: Released

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
-	2021-01-27	Arthur CHEN	Creation of the document
1.0	2021-02-24	Arthur CHEN	First official release

Contents

Figure Index

Table Index

1 Introduction

Quectel LTE standard EC2x series and EG25-G 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, see ***document [1]***.

This document introduces the BT function of the Quectel LTE standard EC2x series and EG25-G modules that should be used in combination with Quectel FC20 series and FC21 modules so as to realize device interconnection through current wireless technology with the lowest power consumption.

Bluetooth (BT) is a wireless technology standard used for exchanging data between fixed and mobile devices over short distances using short-wavelength UHF radio waves in the industrial, scientific and medical radio bands. BT includes traditional Bluetooth and low energy Bluetooth. This document will focus on the implementation of the BLE, SPP and HFP functions in the Bluetooth protocol stack on Quectel EC2x series and EG25-G modules.

Bluetooth Low Energy (Bluetooth LE or BLE) is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries. Compared to Classic Bluetooth, Bluetooth Low Energy is intended to considerably reduce power consumption and cost while maintaining a similar communication range.

The development of classic Bluetooth is based on SPP protocol, which intends to establish a transmission channel between the local Bluetooth device and the remote Bluetooth device to realize data interaction.

HFP stands for hands-free protocol and is a type of Bluetooth used to make voice calls, such as answer, hang up, or reject a call and perform voice calls. HFP defines audio gateway (AG) role and hands-free (HF*) role: HF is the remote audio inputting/outputting mechanism for the audio gateway and provides several remote-control functionalities, which are generally used as the car Bluetooth; AG is the input/output gateway of an audio device, which is generally used for mobile phones. Currently this document will introduce AG role related functionalities.

1.1. Applicable Modules

Table 1: Applicable Modules

Module Series	Module
EC2x series	EC25 series
	EC21 series
	EC20 R2.1
EG25-G	EG25-G

1.2. Special Mark

Table 2: Special Mark

Mark	Definition
*	When an asterisk (*) is used after a function, feature, interface, pin name, AT command, or argument, it indicates that the function, feature, interface, pin name, AT command, or argument is under development and currently not supported, unless otherwise specified.

2 BT Configuration Flowchart

2.1. BLE Configuration Flowchart

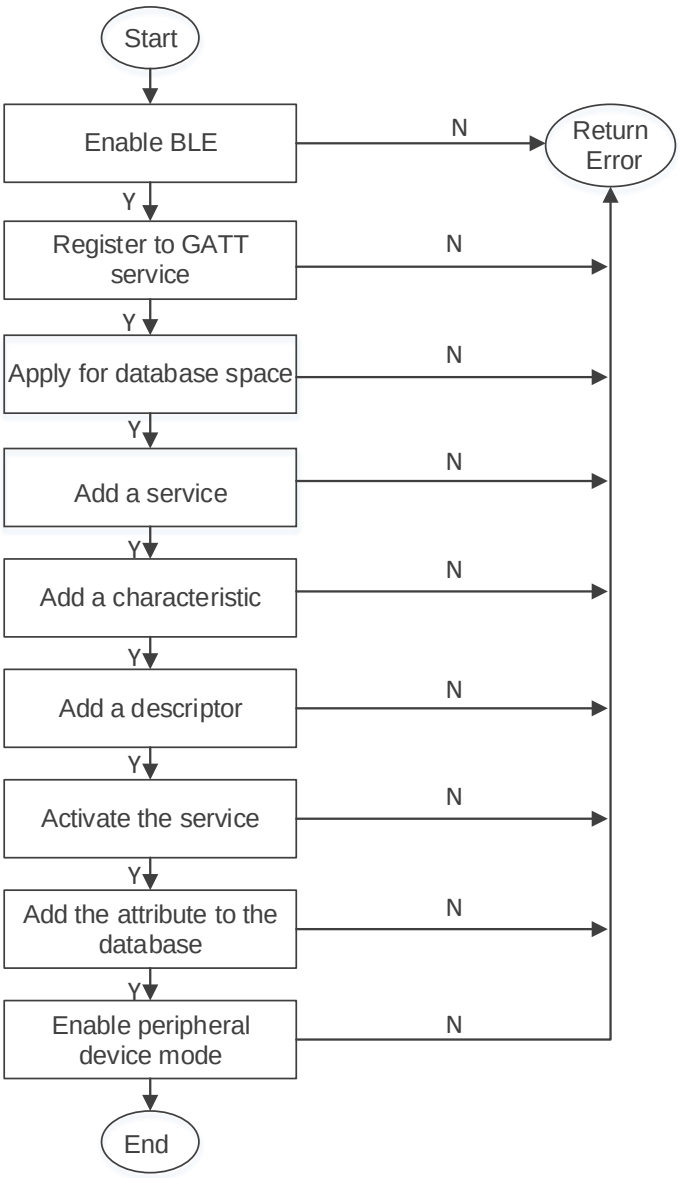

Figure 1: BLE Flowchart

2.2. SPP Configuration Flowchart

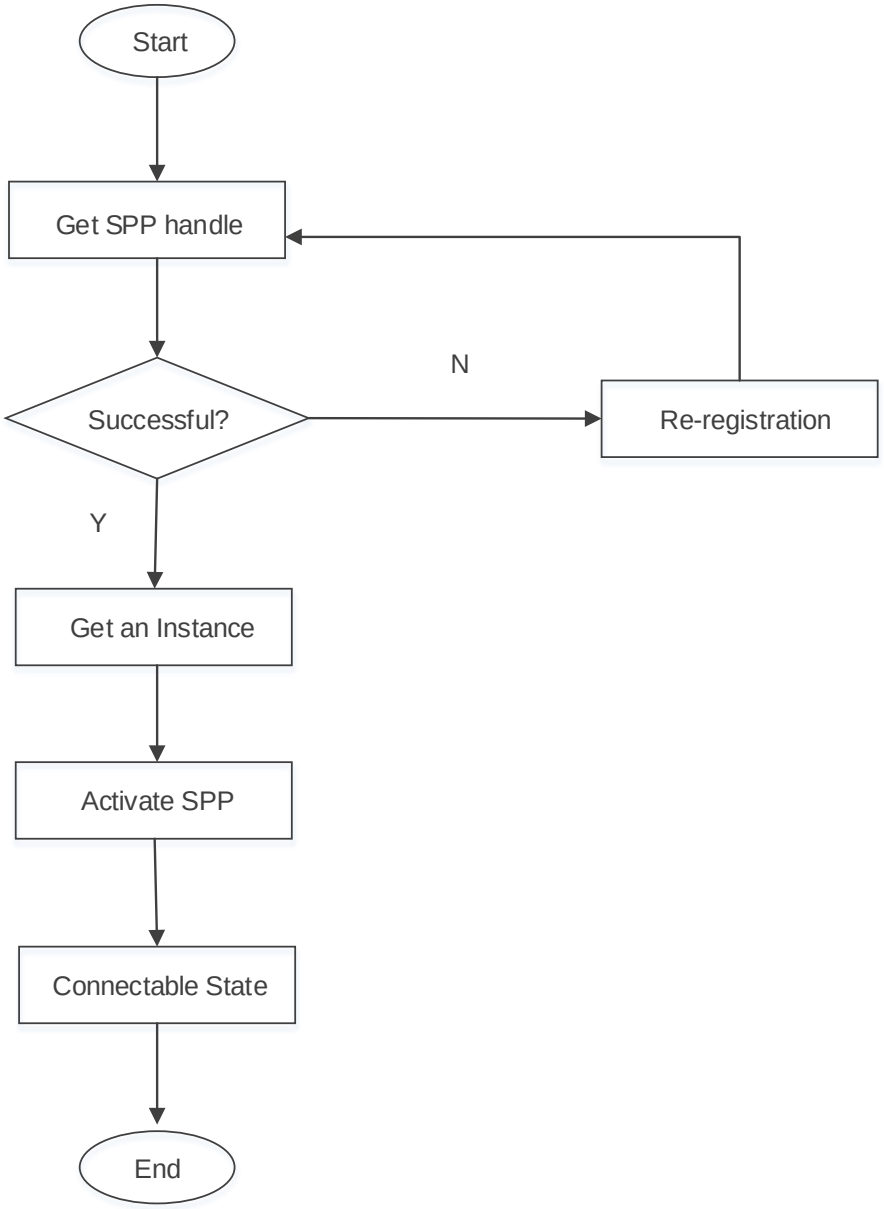

Figure 2: SPP Flowchart

2.3. AG Configuration Flowchart

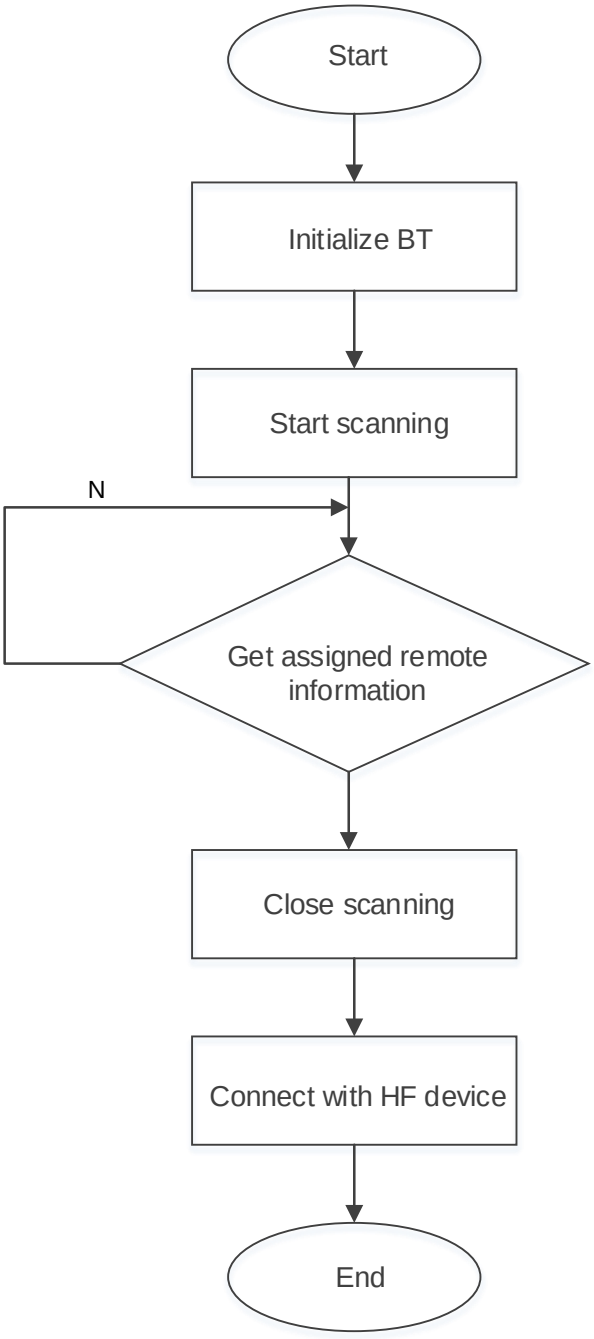

Figure 3: AG Flowchart

3 BT APIs

3.1. BLE APIs

3.1.1. ql_ble_power_on

This function powers on the BLE service.

- **Prototype**

```
int ql_ble_power_on();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.2. ql_ble_power_off

This function powers off the BLE service.

- **Prototype**

```
int ql_ble_power_off();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.3. ql_ble_client_init

This function registers the user callback interface before configuring GATT.

- **Prototype**

```
int ql_ble_client_init(int (*client_cb)(QuecBtPrim type, char *data, int len));
```

- **Parameter**

client_cb:
[In] Process the event sent from the server and give response.

type:
[In] Received event type.

data:
[In] Received data.

len:
[In] Data length.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.4. ql_ble_client_deinit

This function deinitializes the BLE client.

- **Prototype**

```
int ql_ble_client_deinit();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.5. ql_ble_set_local_name

This function sets the device name.

- **Prototype**

```
int ql_ble_set_local_name(char *name);
```

- **Parameter**

name:
[In] The device name.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.6. ql_ble_gatt_register

This function registers to the GATT service to get GATT IDs.

- **Prototype**

```
int ql_ble_gatt_register(QuecBtGattId *gattId);
```

- **Parameter**

gattId:
[Out] The registered GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.7. ql_ble_gatt_unregister

This function deregisters from the GATT service.

- **Prototype**

```
int ql_ble_gatt_unregister(QuecBtGattId *gattId);
```

- **Parameter**

gattId:
[In] The GATT ID to be deregistered.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.8. ql_ble_db_service_add

This function adds a service to the GATT service.

- **Prototype**

```
int ql_ble_db_service_add(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUuid16 uuid, QuecBtUint8 isPrimary);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] The service ID to be added.

uuid:
[In] A service UUID.

isPrimary:
[In] Specifies the service that is a primary, secondary or a referenced service by other services.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.9. ql_ble_db_service_del

This function deletes the added service in the GATT service.

- **Prototype**

```
int ql_ble_db_service_del(QuecBtGattId gattId, QuecBtUint16 svrID,);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] The service ID to be deleted.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.10. ql_ble_db_charact_add

This function adds a characteristic.

- **Prototype**

```
int ql_ble_db_charact_add(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUint16 charactID, QuecBtUint16 uuid, QuecBtUint16 valueLength, QuecBtUint8 prop, QuecBtUint16 attrValueFlags, QuecBtUint8 *value);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] A service ID.

charactID:
[In] The characteristic ID to be added.

uuid:
[In] A characteristic UUID.

valueLength:
[In] Characteristic length.

prop:
[In] Characteristic properties. Different values represent different properties.

attrValueFlags:
[In] Attribute Value Flags. Defines how the Characteristic Value can be accessed.

value:
[In] Characteristic value.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.11. ql_ble_db_charact_del

This function deletes the added characteristic.

- **Prototype**

```
int ql_ble_db_charact_del(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUint16 charactID);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] A service ID.

charactID:
[In] The characteristic ID to be deleted.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.12. ql_ble_db_descriptor_add

This function adds a descriptor.

- **Prototype**

```
int ql_ble_db_descriptor_add(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUint16 charactID, QuecBtUint16 descID, QuecBtUuid16 uuid, QuecBtUint16 valueLength, QuecBtUint8 prop, QuecBtUint16 attrValueFlags, QuecBtUint8 *value);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] A service ID.

charactID:
[In] A characteristic ID.

descID:
[In] The descriptor ID to be added.

uuid:
[In] A descriptor UUID.

valueLength:
[In] The descriptor length.

prop:
[In] Descriptor properties. Different values represent different properties.

attrValueFlags:
[In] Attribute value flags, defines how the characteristic value can be accessed.

value:
[In] Descriptor value.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.13. ql_ble_gatt_descriptor_del

This function deletes the added descriptor.

- **Prototype**

```
int ql_ble_db_descriptor_del(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUint16 charactID, QuecBtUint16 descID);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] A service ID.

charactID:
[In] The deleted characteristics ID.

descID:
[In] Descriptor ID to be deleted.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.14. ql_ble_gatt_send_indication

This function sends an indication.

- **Prototype**

```
int ql_ble_gatt_send_indication(QuecBtGattId gattId, QuecBtConnId connId, QuecBtUint16 attrHandle, QuecBtUint16 valueLength, QuecBtUint8 *value);
```

- **Parameter**

gattId:
[In] GATT ID.

connId:
[In] BLE connected ID.

attrHandle:
[In] Attribute handles.

valueLength:
[In] Length of the indication.

value:
[In] Content of the indication.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.15. ql_ble_gatt_send_notification

This function sends a notification.

- **Prototype**

```
int ql_ble_gatt_send_notification(QuecBtGattId gattId, QuecBtConnId connId, QuecBtUint16 attrHandle, QuecBtUint16 valueLength, QuecBtUint8 *value);
```

- **Parameter**

gattId:
[In] GATT ID.

connId:
[In] BLE connected ID.

attrHandle:
[In] Attribute handle.

valueLength:
[In] Length of the notification.

value:
[In] Content of the notification.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.16. ql_ble_gatt_read_response

This function responds to the request of reading data.

- **Prototype**

```
int ql_ble_gatt_read_response(QuecBtGattId gattId, QuecBtConnId connId, QuecBtUint16 attrHandle, QuecResultCode result, QuecBtUint16 valueLength, QuecBtUint8* value);
```

- **Parameter**

gattId:
[In] GATT ID.

connId:
[In] BLE connected ID.

attrHandle:
[In] Attribute handles.

result:
[In] Return value.

valueLength:
[In] Return value length.

value:
[In] Response content.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.17. ql_ble_gatt_write_response

This function responds to the written data.

- **Prototype**

```
int ql_ble_gatt_write_response(QuecBtGattId gattId, QuecBtConnId connId, QuecBtUint16 attrHandle, QuecResultCode result);
```

- **Parameter**

gattId:
[In] GATT ID.

connId:
[In] BLE connected ID.

attrHandle:
[In] Attribute handles.

result:
[In] Return value.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.18. ql_ble_adverting_start

This function activates advertising.

- **Prototype**

```
int ql_ble_adverting_start(QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.19. ql_ble_adverting_stop

This function stops advertising.

- **Prototype**

```
int ql_ble_adverting_stop(QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.20. ql_ble_gatt_set_adverting_param

This function sets advertising interval.

- **Prototype**

```
int ql_ble_gatt_set_adverting_param(QuecBtGattId gattId, QuecBtUint16 advIntervalMin, QuecBtUint16 advIntervalMax);
```

- **Parameter**

gattId:
[In] GATT ID.

advIntervalMin:
[In] The minimum advertising interval. Range: 32–16384. Unit: 0.625 ms; Default: 256.

advIntervalMax:
[In] The maximum advertising interval. Range: 32–16384. Unit: 0.625 ms; Default: 512.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.21. ql_ble_gatt_disconnect

This function disconnects from the GATT service.

- **Prototype**

```
int ql_ble_gatt_disconnect(QuecBtGattId gattId, QuecBtConnId connId);
```

- **Parameter**

gattId:
[In] GATT ID.

connId:
[In] BLE connected ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.22. ql_ble_gatt_peripheral

This function starts peripheral device mode.

- **Prototype**

```
int ql_ble_gatt_peripheral(QuecBtGattId gattId, QuecBtTypedDeviceAddr addr, QuecBtGattConnFlags flags, QuecBtUint16 mtu, QuecBtConnId *connId);
```

- **Parameter**

gattId:
[In] GATT ID.

addr:
[In] The device address.

flags:
[In] Connecting flags.

mtu:
[In] Maximum transmission unit notified to the remote device during connection establishment.

connId:
[In] BLE connected ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.23. ql_ble_gatt_db_alloc

This function applies for database space.

- **Prototype**

```
Int ql_ble_gatt_db_alloc (QuecBtGattId gattId, QuecBtUint16 numOfAttrHandles, QuecBtUint16 preferredStartHandle);
```

- **Parameter**

gattId:
[In] GATT ID.

numOfAttrHandles:
[In] Number of attribute handles.

preferredStartHandle:
[In] The preferred handle.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.24. ql_ble_gatt_db_dealloc

This function releases the database space.

- **Prototype**

```
int ql_ble_gatt_db_dealloc (QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.25. ql_ble_db_service_set_active

This function is used to activate the service in the database.

- **Prototype**

```
int ql_ble_db_service_set_active(QuecBtGattId gattId, QuecBtUint16 svrID, QuecBtUint8 isActive);
```

- **Parameter**

gattId:
[In] GATT ID.

svrID:
[In] The service ID.

isActive:
[In] The status of activation.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.26. ql_ble_gatt_db_add

This function adds the service to the database.

- **Prototype**

```
int ql_ble_gatt_db_add(QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.27. ql_ble_gatt_db_remove

This function deletes the database.

- **Prototype**

```
int ql_ble_gatt_db_remove(QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.1.28. ql_ble_db_show

This function prints the database information in stdout.

- **Prototype**

```
int ql_ble_db_show(QuecBtGattId gattId);
```

- **Parameter**

gattId:
[In] GATT ID.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2. SPP APIs

3.2.1. ql_spp_power_on

This function powers on the BT server.

- **Prototype**

```
int ql_spp_power_on();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.2. ql_spp_power_off

This function powers off the BT server.

- **Prototype**

```
int ql_spp_power_off();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.3. ql_spp_client_init

This function initializes the SPP client.

- **Prototype**

```
int ql_spp_client_init();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.4. ql_activate_spp

This function activates Quectel FC20 series or FC21 module as an SPP slave device and waits the terminal device to connect.

- **Prototype**

```
int ql_activate_spp();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.5. ql_deactivate_spp

This function deactivates SPP.

- **Prototype**

```
int ql_deactivate_spp();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.6. ql_connect_spp

This function activates SPP as master device and connect it with other SPP devices.

- **Prototype**

```
int ql_connect_spp(QuecDeviceAddr * addr);
```

- **Parameter**

addr:
[In] Slave device MAC address.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.7. ql_write_spp

This function sends data to remote SPP device.

- **Prototype**

```
int ql_write_spp(QuecBtUint16 valutLength, QuecBtUint8 * value);
```

- **Parameter**

valueLength:
[In] To be sent data length.

value:
[In] To be sent data.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.2.8. ql_disconnect_spp

This function actively disconnects from SPP.

- **Prototype**

```
int ql_disconnect_spp();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3. AG APIs

3.3.1. ql_hfg_power_on

This function powers on the BT server.

- **Prototype**

```
int ql_hfg_power_on();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.2. ql_ble_power_off

This function powers off the BT server. See ***Chapter 3.1.2*** for details.

- **Prototype**

```
int ql_ble_power_off();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.3. ql_ble_client_init

This function initializes BLE to be an AG device. See ***Chapter 3.1.3*** for details.

- **Prototype**

```
int ql_ble_client_init(int (*client_cb)(QuecBtPrim type, char *data, int len));
```

- **Parameter**

client_cb:
[In] Process the event sent from the server and give response.

type:
[In] Received event types.

data:
[In] Received data.

len:
[In] Data length.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.4. ql_open_scan_device

This function opens BT device scanning.

- **Prototype**

```
int ql_open_scan_device();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.5. ql_close_scan_device

This function closes devices scanning.

- **Prototype**

```
int ql_close_scan_device();
```

- **Parameter**

N/A

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.6. ql_hfg_connect

This function connects with the HF device.

- **Prototype**

```
int ql_hfg_connect(QuecBtDeviceAddr address);
```

- **Parameter**

address:
[In] The MAC address of the HF device.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

3.3.7. ql_hfg_cancelconnect

This function disconnects with the HF device.

- **Prototype**

```
int ql_hfg_cancelconnect(QuecBtDeviceAddr address) ;
```

- **Parameter**

address:
[In] The MAC address of the HF device.

- **Return Value**

0	Indicates the operation is successful
Other values	Indicates the operation is unsuccessful

4 Demonstration Steps of BT Use

4.1. Demonstration Steps of BLE Configuration

Step 1: Execute the following script to enable GATT service.

```
int ql_ble_power_on();
```

Step 2: The following function is called to initialize the client. The callback function *client_cb* is registered by the user and is used to parse the message sent by the server.

```
ql_ble_client_init(client_cb);
```

Step 3: Call the following function to register to the GATT service to obtain the GATT ID. Store the registered ID in the address space specified by *gattId*.

```
ql_ble_gatt_register(&gattId);
```

Step 4: Call the following function to apply for a database space which is used to store attribute descriptors.

```
ql_ble_gatt_db_alloc(gattId, numOfAttrHandles, preferredStartHandle);
```

Step 5: Call the following function to add a service, and then add characteristics and descriptors based on the service.

```
ql_ble_db_service_add(gattId, 1, QUEC_BT_GATT_UUID_DEVICE_INFORMATION_SERVICE, 1);
```

Step 6: Call the following function to add a characteristic based on the services.

```
ql_ble_db_charact_add(gattId,1,1, QUEC_BT_GATT_UUID_MANUFACTURER_NAME_STRING_CHARAC,128, QUEC_ATT_PERM_READ | QUEC_ATT_PERM_INDICATE, QUEC_BT_GATT_ATTR_FLAGS_IRQ_READ, name);
```

Step 7: Call the following function and add a descriptor based on the services and their characteristics.

```
ql_ble_db_descriptor_add(gattId, 1, 1, 1, 0x2902, 2, QUEC_ATT_PERM_READ, QUEC_BT_GATT_ATTR_FLAGS_NONE, (QuecBtUint8 *)&value);
```

Step 8: Call the following function to activate the service, that is, to register the service added in the above steps to the server.

```
ql_ble_db_service_set_active(gattId, 1, 1);
```

Step 9: Call the following function to add services to the database, that is, to the applied database space.

```
ql_ble_gatt_db_add(gattId);
```

Step 10: Call the following function to set the device to peripheral device mode and wait for connecting.

```
ql_ble_gatt_peripheral(gattId, t_addr, QUEC_BT_GATT_FLAGS_UNDIRECTED, 0, &connId);
```

4.2. Demonstration Steps of SPP Configuration

4.2.1. Module as a Slave Device

Step 1: Call the following function to turn on the BT server.

```
ql_spp_power_on();
```

Step 2: The following function is called to initialize the SPP client.

```
ql_spp_client_init();
```

Step 3: The following function is called to activate the SPP as slave device.

```
ql_activate_spp();
```

4.2.2. Module as a Master Device

Step 1: Call the following function to power on the BT server.

```
ql_spp_power_on();
```

Step 2: The following function is called to initialize the SPP client.

```
ql_spp_client_init();
```

Step 3: The following function is called to activate the SPP as master device.

```
ql_connect_spp();
```

4.3. Demonstration Steps of AG Configuration

Step 1: Call the following function to power on the BT server.

```
ql_hfg_power_on()
```

Step 2: Call the following function to initialize BLE to be an AG device.

```
ql_ble_client_init(client_cb_func)
```

Step 3: Call the following function to scanning the BT devices.

```
ql_open_scan_device()
```

Step 4: Call the following function to stop scanning the BT devices.

```
ql_close_scan_device();
```

Step 5: Call the following function to connect with the HF device.

```
ql_hfg_connect(remote_device_addr);
```

5 Appendix A References

Table 3: Related Document

SN	Document Name	Description
[1]	Quectel_EC2x&EG9x&EG25-G_Series_QuecOpen_Quick_Start_Guide	Quick start guide for QuecOpen solution of EC2x series, EG9x series and EG25-G modules

Table 4: Terms and Abbreviations

Abbreviation	Description
AG	Audio Gateway
API	Application Programming Interface
BLE	Bluetooth Low Energy
BT	Bluetooth
GATT	Generic Attribute Profile
HF	Hands Free
HFP	Hands-free Profile
ID	Identity
IoT	Internet of Things
LTE	Long-Term Evolution
SPP	Serial Port Profile
UHF	Ultra High Frequency
UUID	Universally Unique Identifier