QN902x OTA Programming Guide

Rev. 1.3 — 04 April 2018 Application note

Document information

Info	Content
Keywords	OTA Server, OTA Client, API, Android, IOS
Abstract	The OTA is used to upgrade the firmware of QN902x over the air.

Revision history

Rev	Date	Description
0.1	20140519	Initial release
0.2	20140704	Add OTA control functions
0.3	20141017	Add CFG_OTAS_APP_CTRL otas_change_service_uuid()
0.4	20141107	OTA support update data file
1.0	20150330	Updated by merging programming, integration with IOS and Android in one doc and migrate to NXP template
1.1	20150817	Update the encryption part
1.2	20150024	Reviewed by PSP
1.3	20180404	File location change for Android, some general changes

Contact information

For more information, please visit: http://www.nxp.com

Contents

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

Over-The-Air programming (OTA), is used to upgrade the firmware of QN902x over the air. This document, described with code examples, on how to implement the OTA application as an OTA server in BLE peripherals and an OTA application as OTA client in Android/IOS devices.

As described in the *OTA Profile Guide,* the profile defines two roles: OTA Server and OTA Client.

- The OTA Server shall be a GATT server.
- The OTA Client shall be a GATT client.

The **Figure 1** shows the relationships between services and the two profile roles.

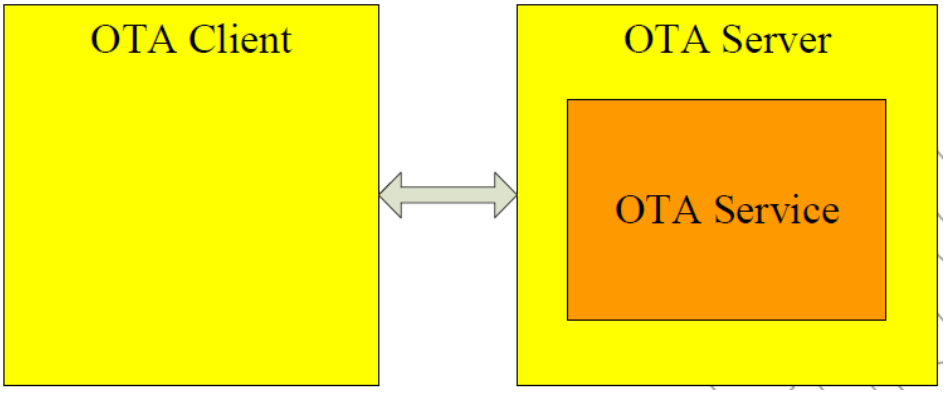

Figure 1 Role / Service Relationships

Note: Profile roles are represented by yellow boxes and services are represented by orange boxes. An OTA Server shall instantiate one and only one OTA Service.

OTA clients for Android and IOS can be downloaded from Collabnet which is an NXP customer support site.

2. OTA Server in QN9020

2.1 Project

The project can be opened with the following IAR and KEIL workspace files:
C:\QBlue\QN9020\QBlue-X.X.X\Projects\BLE\prj_ota\iar\ota.eww
C:\QBlue\QN9020\QBlue-X.X.X\Projects\BLE\prj_ota\keil\ota.uvproj

2.2 Software Description

The OTA application is implemented in the following files:

- otas_task.h: Application function
- qn_ota.lib: OTA Profile

2.2.1 User Configuration

To support OTA feature, the following macros shall be defined in the 'usr_config.h'.

- #define CFG_PRF_OTAS
- #define CFG_TASK_OTAS TASK_PRF7 (Mandatory)

The following macro shall be defined in the 'app_config.h'.

- #define ENAB_OTAS_APP_CTRL (Optional : enable App control OTA feature relevant code)
- #define ENAB_OTAS_SET_UUID (Optional : enable App change OTA service UUID feature relevant code)
- #define ENAB_OTAS_SEND_DATA (Optional : enable OTA service set data addr and send data)

2.2.2 Initialization

The initialization of the application occurs in following phases:

Step1: The **otas_init(uint32_t fw2_start_addr, enum ota_crypt_t crypt, const uint8_t key[16])** function is called by the profiles register function(**prf_init_reg(prf_init)**). This function registers OTA task into kernel.

- **fw2_start_addr:** firmware 2 start address – Default: 0x12000
- **crypt:** Enable or disable encryption - Default: OTA_ENABLE_ENCRYPT
- **key[16]:** AES 128 key (16bytes) - Default: 0x112233445566778899 00AABBCCDDEEFF

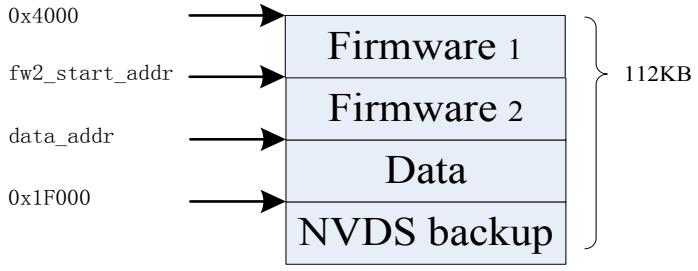

Figure 2 Flash Layout

Step2: Following function is called by the **app_create_server_service_DB()** function.

- **app_otas_create_db():** This function is used to create server service database.

2.2.3 Optional Initialization

Following initialization for customer to control OTA profile. These are not essential.

App control OTA start (optional)

Step1: Function **otas_control(struct otas_ctrl_info *pctrl_info)** must be called after otas_init(…).

Step2: App received msg **OTAS_TRANSIMIT_STATUS_IND.** Check the msg parameter - the

OTA status whether it is **OTA_STATUS_START_REQ**. If so, Function **app_ota_ctrl_resp(enum otas_ctrl_resp)** is called to start OTA.

App change OTA service UUID (optional)

Step 1: Function **app_otas_change_svc_uuid(uint8_t *p_uuid)** must be called before app_otas_create_db().

- **p_uuid:** pointer to a 128bit new UUID.

App set data start address (optional)

Step 1: Function **app_otas_set_data_addr(uint32_t data_addr)** must be called before app_otas_create_db().

- **data_addr:** start address in flash to put data.

If user doesn't call this function, the default data_addr is 0x1F000, and therefore the OTA will not support data upgrade

2.3 API

2.3.1 otas_init ()

Prototype:

Enum ota_status_t otas_init (uint32_t fw2_start_addr, enum ota_crypt_t crypt, const uint8_t key[16])

Description:

This function performs all the initializations of the OTA module.

Parameters:

Parameters	Desciption
fw2_start_addr	Set second zone start address.
crypt	OTA_ENABLE_ENCRYPT: Enable encryption OTA_DISABLE_ENCRYPT: Disable encryption.
key[16]	AES 128 key (16bytes).

Returns:

OTA_STATUS_OK: The OTAS initialized successfully.

OTA_STATUS_FW2_ADDR_INVALID: The OTAS initialization failed; firmware address is invalid.

OTA_STATUS_DEVICE_NOT_SUPPORT_OTA: The OTAS initialization failed; the device does not support OTA.

2.3.2 app_otas_create_db()

Prototype:

void app_otas_create_db(void)

Description:

This function is used to create OTA server service database.

2.3.3 app_otas_enable_req()

Prototype:

void app_otas_enable_req (uint16_t conhdl, uint8_t sec_lvl)

Description:
This function is used for enabling the Reporter role of the OTA profile. After calling this function, the services are unhidden from the peer device discovery.

Parameters:

Parameters	Description
conhdl	Connection handle for which the OTA Server role is enabled
sec_lvl	Security level required for protection of attributes. Service Hide and Disable are not permitted. Possible values are: PERM_RIGHT_ENABLE : Enable access PERM_RIGHT_UNAUTH : Access Requires Unauthenticated link PERM_RIGHT_AUTH : Access Requires Authenticated link

Returns:
None

2.3.4 otas_control ()

Prototype:
void otas_ control(struct otas_ ctrl_info *pctrl_info)

Parameters:

Parameters	Description			
pctrl_info	Struct otas_ ctrl_info data field:	uint8_t	ctrl_flag	0: ota start, control by profile 1: ota start ,control by app
		uint8_t	reserved	reserved for future , must be set to 0x00

Returns:
None.

2.3.5 app_ota_ctrl_resp ()

Description:

Response the OTA data transfer to start or reject.

This will transfer Msg OTAS_CONTRL_APP_RESP to OTA server to start or stop data transfer.

This function only works when the OTA status is OTA_STATUS_START_REQ. Before calling this function to start OTA, other peripherals impacting flash write should be stopped, such as ADC...etc.

Prototype:
void app_ota_ ctrl_resp(enum otas_ctrl_resp)

Parameters:

Parameters	Description
ctrl_resp	enum otas_ctrl_resp data field: START_OTA — old way, control by profile REJECT_OTA — new way, control by app

Response:

None

2.3.6 app_otas_change_svc_uuid ()

Description:

Function to change OTA service UUID to user defined. This function must be called before OTA creates a database.

Prototype:

uint8_t app_otas_change_svc_uuid (uint8_t *p_uuid);

Parameters:

uint8_t *	p_uuid	Pointer to a 128bit OTA service UUID

Response:

true : change to new UUID.

false : p_uuid pointer is NULL.

2.3.7 app_otas_set_data_addr()

Description:

Function to change OTA data address in flash. This function must be called before OTA creates a database

Prototype:

uint8_t app_otas_set_data_addr(uint32_t data_addr);

Parameters:

uint8_t *	data_addr	Pointer to a 128bit OTA service UUID

Response:

true : Success to set data addr

false : Fail to set data address because one of following reasons :

1. data_addr > 0x1F000(Flash limited)
2. data_addr < firmware 2 start address
3. data_addr is not an integer of 4k.

2.3.8 otas_get_app_info ()

Description:

Get available flash block information, which will be used for placing a new version of the application.

Prototype:

uint8_t otas_get_app_info(struct otas_app_information_t *ota_app_information);

Parameters:

struct otas_app_information_t *	data_addr	Pointer to get the app information.

Response:

true : - Success to get the information

false : - program do not support OTA

- program is in the status of OTA communication,

2.4 Msg Interface

2.4.1 OTAS_TRANSIMIT_STATUS_IND

Parameters:

struct	otas_transimit_status_ind	Data field : uint8_t, status, Status: OTA_STATUS_START_REC, OTA_STATUS_ONGOING, OTA_STATUS_FINISH_OK, OTA_STATUS_FINISH_FAIL, uint16_t, status_des, status detail description information : - Total size - Received bytes - Error type - NULL

Description:

Status of OTA transmission indication to app.

In the OTA_STATUS_START_REQ status, app can send msg to control OTA.

The relationship of status and status description:

Status	**status detail description information**
OTA_STATUS_START_REQ	Total size
OTA_STATUS_ONGOING	Received bytes
OTA_STATUS_FINISH_OK	NULL
OTA_STATUS_FINISH_FAIL	Error type

Error type:

Include following 5 types errors.

Status	**status detail description information**
0x01	Current packet checksum error
0x02	current packet length overflow or equal to 0
0x03	Device doesn't support OTA
0x04	OTA firmware size overflow or equal to 0
0x05	OTA firmware verify error

3. OTA Client Overview

The libqblueOta library acts as OTA client role, which is used by application to upgrade the firmware of the OTA server.

3.1 Features

- Prevent injection and impersonation attack.
- Protect data security over air.
- Firmware recoverable if upgrade failed.
- Support resume after disconnection. This means that once the connection is broken during an upgrade process, it will just re-connect and continue downloading firmware without needing to start over from the beginning.

Note: In order to guarantee the upgrade success, please make sure the firmware run-time size (including Code, RO-data, ZI-data, RW-data) is less than 50K bytes.

3.2 Overview

The OTA Client Diagram consists of three parts:

App layer:

- Send requests to **CoreBluetooth** layer and use OTA API method.
- Update OTA application UI.
- Response exceptions from **CoreBluetooth** layer.

API Layer:

- Receive and process App commands from App layer.
- Send requests to **CoreBluetooth** layer.
- Update OTA status to App layer.

CoreBluetooth Layer:

- Receive and response requests from App layer and API layer.
- Process connections' delegate of app layer.
- Update value and state for API layer.

The OTA Diagram of Android app shown in Figure 3:

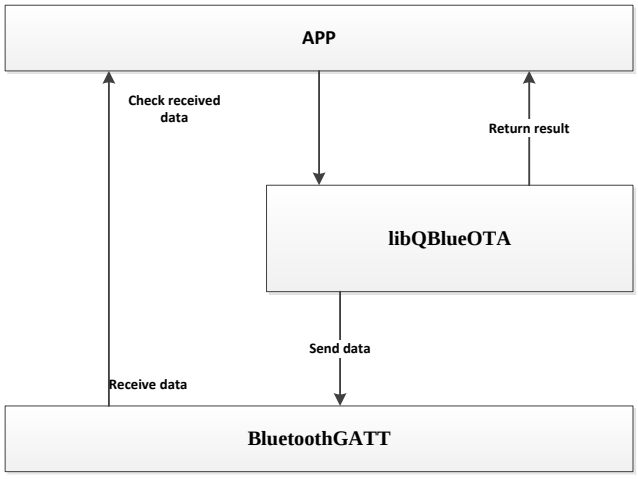

Figure 3 OTA Diagram of Android platform

The OTA Diagram IOS app is shown in Figure 4:

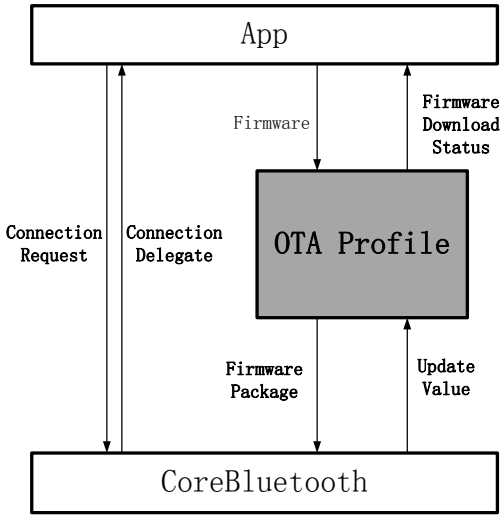

Figure 4 OTA Diagram of IOS Platform

4. OTA Integration in Android

4.1 Flowchart

The OTA general flowchart is the following:

- Scan BLE peripherals nearby.
- Establish a connection.
- Discover OTA services and characteristics.
- Load a firmware file, here you'd use the method: *otaStart*, which starts OTA.
- OTA state machine: start to transmit data to Qn902x side, after sending each package, you can get a result, which includes whether it is sent successful or not. Then you can refresh UI according to these results (In the function *otaGetProcess*).

- The function *otaGetProcess*, will return the process information and the final result of whether the OTA process was success or failure.
- After verify confirmation, send a reset command to reset the BLE peripheral.

OTA flowchart is shown in **Figure 5**:

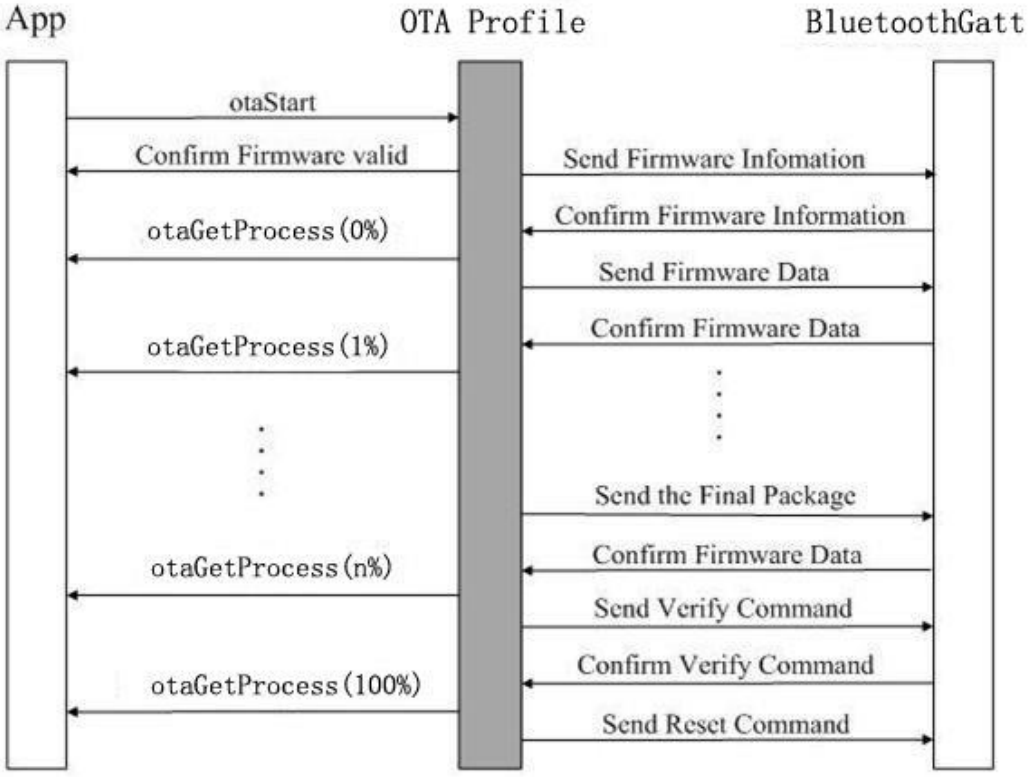

Figure 5 OTA flowchart

4.2 API and Variables Description

There are three public classes in the libqblueota library. Global variables are defined in class classotaGlobalVariables.

Class BluetoothLeInterface is a wrapper of Android GATT APIs.

Class otaManager is key class to OTA transmission .

Their relationship is shown in **Figure 6**.

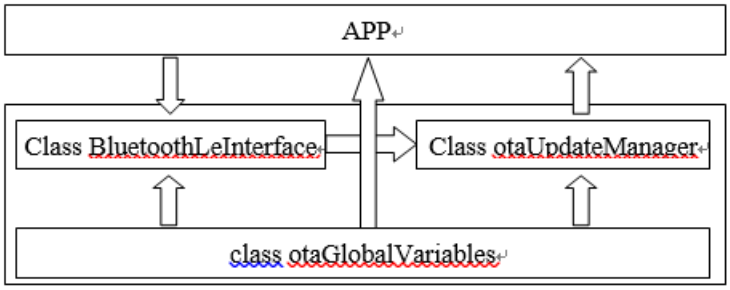

Figure 6 libQblueOTA Library Structure Diagram

4.2.1 Class otaGlobalVariables

The variables defined in class *otaGlobalVariables* are public. It includes UUID definition and Return value definiton. The specification can be found in source code of OTA demo.

4.2.2 Class BluetoothLeInterface

General description

Class *BluetoothLeInterface* defines a group of necessary APIs which are used in LibQblueOta library.
It's a public abstract class, so need to be extended by a child class and created an instance in application.

The class is defined as below:

```
public abstract class BluetoothLeInterface{
   public boolean bleInterfaceInit(BluetoothGatt bluetoothGatt);
   public boolean writeCharacteristic(byte[] data);
   public boolean setCharacteristicNotification(boolean enabled);
};
```

The sample code is as below:

```
public class OtaActivity{
  BluetoothGatt mBluetoothGatt;
  ……
   private class updateInstance extends
   BluetoothLeInterface{
      public boolean bleInterfaceInit(BluetoothGatt
   bluetoothGatt){
         super.bleInterfaceInit(bluetoothGatt);
       }
   }

   updateInstance ins=new updateInstance();
   ins.bleInterfaceInit(mBluetoothGatt); //Ensure
   mBluetoothGatt is available before you call this function
}
```

API Description

1. public boolean bleInterfaceInit(BluetoothGatt bluetoothGatt)

Description:

Initial OTA service, get OTA service and characteristics.

Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler

Returns:

True ble interface initial is successful.
False ble interface initial is failed.

Note:

BluetoothGatt should have been initialized before calling this function.

2. public boolean writeCharacteristic(byte[] data)

Description:

Write characteristic, the default characteristic UUID is in otaGlobalVariables. UUID_OTA_WRITE_CHARACTERISTIC.

Parameters:

In	data	The data to be written

Returns:

True write data is successful.

False write data is failed.

Note:

In Android, this function is asynchronous. User needs check the result in BluetoothGATT callback function

3. public boolean setCharacteristicNotification(boolean enabled)

Description:

Set characteristic notification. The default Characteristic UUID is in otaGlobalVariables: UUID_OTA_NOTIFY_CHARACTERISTIC.

Parameters:

In	enabled	true : enabled false: disabled

Returns:

True successful.

False failed.

4.2.3 Class otaManager

General description

otaManager defines some functions to implement OTA upgrade.

```
public class otaManager{
  public otaResult otaStart(String
file,BluetoothLeInterface intf);
   public void otaStop();
  public otaResult otaGetProcess(int[] extra);
  public void otaGetResult(byte notify_data[]);
  public void notifyWriteDataCompleted();
};
```

API description

public otaResult otaStart(String file, BluetoothLeInterface intf)

Description:

Start OTA upgrade.

Parameters:

In	file	Firmware file path, the path should be absolute.

In	intf	BluetoothLeInterface object

Returns:

OTA_RESULT_INVALID_ARGUMENT parameters are invalid.
OTA_RESULT_SUCCESS start successful.

Note:

None.

See also:

public void otaStop();

public void otaStop()

Description:

This function is used to stop OTA upgrade.

Parameters:

None.

Returns:

None.

See also:

public otaResult otaStart(String file,BluetoothLeInterface intf);

public otaResult otaGetProcess(int[] extra)

Description:

This function is used to get OTA information during upgrade.

Parameters:

Out	extra[]	Integer array to transfer progress information. The size should be >=8. Three elements are used, the remaining is reserved. extra[0] is OTA upgrade percentage; extra[1] is OTA upgrade Byte Rate; extra[2] is OTA upgrade elapsed time;

Returns:

OTA_RESULT_SUCCESS No error is occurred during upgrade
Error Code Error is occurred

Note:

Customer could create a thread to call this function continuously.

public void otaGetResult(byte notify_data[])

Description:

This function is used to get the notify data. The notify data is used to control the OTA upgrade transmission process.

In	notify_data[]	Received data from GATT server.

Parameters:
Returns:
None.

public void notifyWriteDataCompleted()

Description:
This function is used to notify the otaQblueLibrary that write Characteristic successfully.
Parameters:
None.
Returns:
None.
Note:
In the Android system 'Write Characteristic' action is asynchronous. There is a callback named BluetoothGattCallback .onCharacteristicWrite() to notify user the result of writeCharacteristic(). So this function should be invoked in onCharacteristicWrite().

4.3 Integration Note

The detail steps to create the OTA client application:

a) Create *class otaUpdateManager* object.
b) Create *class updateInstance* which extends *class BluetoothLeInterface*.
c) Invoke *updateManager.startUpdate(fwFile,ins)* to start upgrade.
d) Create thread to get process information continuously
e) Invoke *updateManager.otaGetResult(notifyData)* when notify data have been received.
f) Invoke *updateManager.notifyWriteDataCompleted()* when send data is successful.

```
public class OtaActivity{
  BluetoothGatt mBluetoothGatt;
  private shouldStop=false;
   private otaManager updateManager=new otaManager();

   updateInstance ins=new updateInstance();
   //Ensure mBluetoothGatt is available
   ins.bleInterfaceInit(mBluetoothGatt);

   String fwFile=getFirmwareFile();
   updateManager.startUpdate(fwFile,ins);
   updateThread.start();

   private final BluetoothGattCallback mGattCallback = new
   BluetoothGattCallback() {
        public void onConnectionStateChange(BluetoothGatt
     gatt,
           int status, int newState){
         if(newState == BluetoothProfile.STATE_DISCONNECTED)
            shouldStop=true;
        }
         public void onCharacteristicWrite(BluetoothGatt gatt,
                  BluetoothGattCharacteristic
     characteristic,int status){
           if(status==BluetoothGatt.GATT_SUCCESS)
            updateManager.notifyWriteDataCompleted();
            else
            mStopUpdate=true;
         }
        public void onCharacteristicChanged(BluetoothGatt
     gatt,
                     BluetoothGattCharacteristic
     characteristic){
            byte[] notifyData=characteristic.getValue();

            updateManager.otaGetResult(notifyData);
        }
   }

   Thread updateThread=new Runnable(){
    While(!shouldStop){
      Thread.sleep(100);
      if(updateManager. otaGetProcess()){
        shouldStop=true;
        updateManager.stopUpdate();
       }else{
        updateProcessDialog();
       }
   }
};
```

4.4 Example code

There is an example Android project in Collabnet named 'OTA_Android_xxx.zip'. The example shows how to use the libqblueota library to implement OTA client application.

5. OTA Integration in IOS

The libQBlueOta library (xx\Ota\LibOta) acts as OTA client role, which is used by application to upgrade the firmware of the OTA server.

5.1 Flowchart

The OTA general flowchart is the following:

- Scan nearby BLE peripherals.
- Establish a connection.
- Discover OTA services and characteristics.
- Load a firmware file, here you'd use the method: *otaStart*, which starts OTA.
- Ota state machine: start to transmit data to Qn902x side, after implement each package, you can get a result, which includes whether it is sent successful and how many packages have been sent. Then you can refresh UI according to these results (In the delegate *didOtaAppProgress*).
- In the delegate *didOtaAppResult*, it will update the final result to whether the OTA is success or failure.
- After confirm verify, send reset command to reset the BLE peripheral.

OTA flowchart is shown in Figure 7:

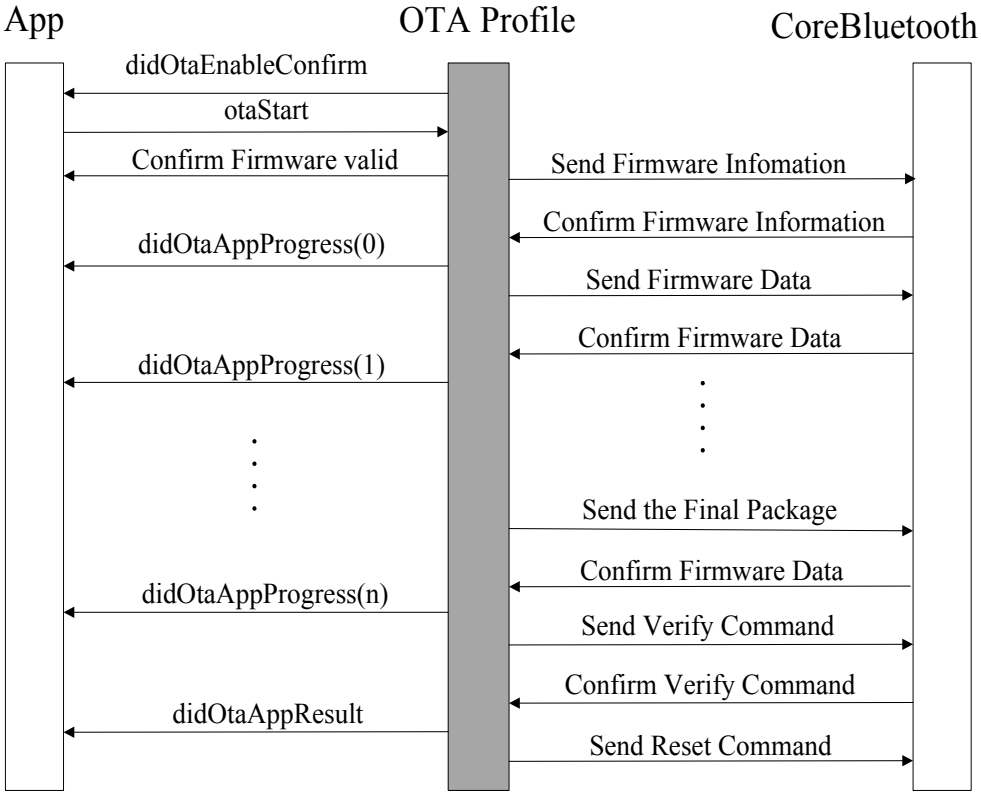

Figure 7 OTA flowchart in IOS

5.2 API and Delegate Description

These functions consist of one API function and four delegate functions. API functions are implemented to register user's UUID and start the OTA process. The delegate functions are implemented to update the status of the current transmitting package and the final OTA update result.

5.2.1 didOtaEnableConfirm ()

Prototype:

(void)didOtaEnableConfirm : (CBPeripheral *)aPeripheral
withStatus : (enum otaEnableResult) otaEnableResult;

Description: Before user sends or receives data, must first check the results.

Parameters:

out	aPeripheral	The target peripheral with OTA profile
out	otaEnableResult	Indicate the device service, character status and configuration of the device. - OTA_CONFIRM_OK: All of following three conditions are satisfied, OTA service is discovered, OtaCharWr is discovered, OTACharNTF is discovered,

		- OTA_CONFIRM_FAILED: service creation is not success

Returns:

None.

5.2.2 otaStart()

Prototype:

-(enum otaApiLoadFileResult)otaStart : (CBPeripheral *)aPeripheral
withDataByte : (const uint8_t *)firmwareAddr
withLength : (uint32_t)firmwareLength
withFlag : (BOOL)fResume;

Description: The function is used by the application to upgrade a firmware file through the OTA client role. After a peripheral has connected and the OTA service/characteristics are discovered, then the API function is called. The OTA state machine will run till completion or if the connection is broken. If the connection is broken the user may resume the connection by calling API fResume with parameter set to TRUE. The upgrade process will resume from where the connection was broken initially.

Parameters:

In	aPeripheral	The target peripheral with OTA profile
In	firmwareAddr	Firmware 2 pointer, points to the firmware file's head
In	firmwareLength	firmware length
In	fResume	whether the connection is resumed or not

Returns:

The return value *OTA_API_FILE_NO_ERROR* means that the firmware file is loaded without error.

The return value *OTA_API_FILE_ERROR* means that the firmware file is loaded with error and OTA do nothing.

5.2.3 didOtaMetaDataResult ()

Prototype: (void)didOtaMetaDataResult : (enum otaResult)otaMetaDataSentStatus;

Description: This function can be used by the OTA application to refresh UI. After OTA transfers the meta-data, the method updates the status up to the app layer. The parameter otaMetaDataSentStatus means the status after the meta-data has been sent out. After 10 seconds timeout, it will output the result OTA_RESULT_DEVICE_NOT_SUPPORT_OTA.

Parameters:

Out	otaMetaDataSentStatus	The status of transmission meta data: OTA_RESULT_SUCCESS , OTA_RESULT_DEVICE_NOT_SUPPORT_OTA

Returns: None

5.2.4 didOtaAppProgress()

Prototype:

(void) didOtaAppProgress : (enum otaApiResult)otaPackageSentStatus
withSentBytes : (uint16_t)otaDataSent;

Description: The function can be used by OTA application to refresh UI. After OTA transfers each package of data, the method updates the status to the app layer. The parameter otaDataSent means how many bytes have been sent out, it can be used to calculate data rate or progress status.

Parameters:

out	otaPackageSentStatus	The status of transmission current package: OTA_RESULT_SUCCESS , OTA_RESULT_PKT_CHECKSUM_ERROR, OTA_RESULT_PKT_LEN_ERROR, OTA_RESULT_DEVICE_NOT_SUPPORT_OTA, OTA_RESULT_FW_SIZE_ERROR, OTA_RESULT_FW_VERIFY_ERROR,
out	otaDataSent	data sent in bytes

Returns:

None

5.2.5 didOtaAppResult()

Prototype:

(void) didOtaAppResult : (enum otaResult) otaResult;

Description: This function can be used by the OTA application to refresh UI when the OTA finishes. For example, if the OTA fails, then the user can reload the firmware and start the OTA process again. Or check the hardware, re-connect and re-start.

Parameters:

out	otaResult	The OTA final result OTA_RESULT_SUCCESS , OTA_RESULT_PKT_CHECKSUM_ERROR, OTA_RESULT_PKT_LEN_ERROR, OTA_RESULT_FW_VERIFY_ERROR,

Returns:

None.

5.3 Integration Note

a) Please insert the "*bleDidUpdateCharForOtaService*"delegate method in the *didDiscoverCharacteristicsForService* delegate. The delegate is to update write characteristic and notify characteristic for OTA service.

```
- (void) peripheral : (CBPeripheral *)aPeripheral
   didDiscoverCharacteristicsForService : (CBService *)service error : (NSError *)error
{
   /// for profile delegate
   [bleUpdateForOtaDelegate bleDidUpdateCharForOtaService : aPeripheral
```

```
                withService : service
                  error : error];

    /// user code
    ……

}
```

b) Please insert the "***bleDidUpdateValueForOtaChar***" delegate method in the "***didUpdateValueForCharacteristic***" delegate. The delegate is to update value for notification characteristic.

```
- (void) peripheral:(CBPeripheral *)aPeripheral
didUpdateValueForCharacteristic:(CBCharacteristic *)characteristic error:(NSError
*)error
{
   for (CBService *aService in aPeripheral.services)
   {
      [bleUpdateForOtaDelegate bleDidUpdateValueForOtaChar : aService
                              withChar : characteristic
                                error : error];
      /// user code
      ……
   }
}
```

5.4 Example code

There is an example iOS project named 'OTA_IOS_xxx.zip' in Collabnet and it can be found in "xx\Ota\". It shows how to use the libQBlueOta library to implement firmware upgrade.

6. Download and Upgrade

If you want to use OTA to upgrade an application, first download the bin file using the ISP tool with OTA option checked in the ISP tool. Then install the application developed as an OTA client in Android or IOS platform of choice.

6.1 ISP Download

Use ISP tool to download OTA bin file with OTA option checked.

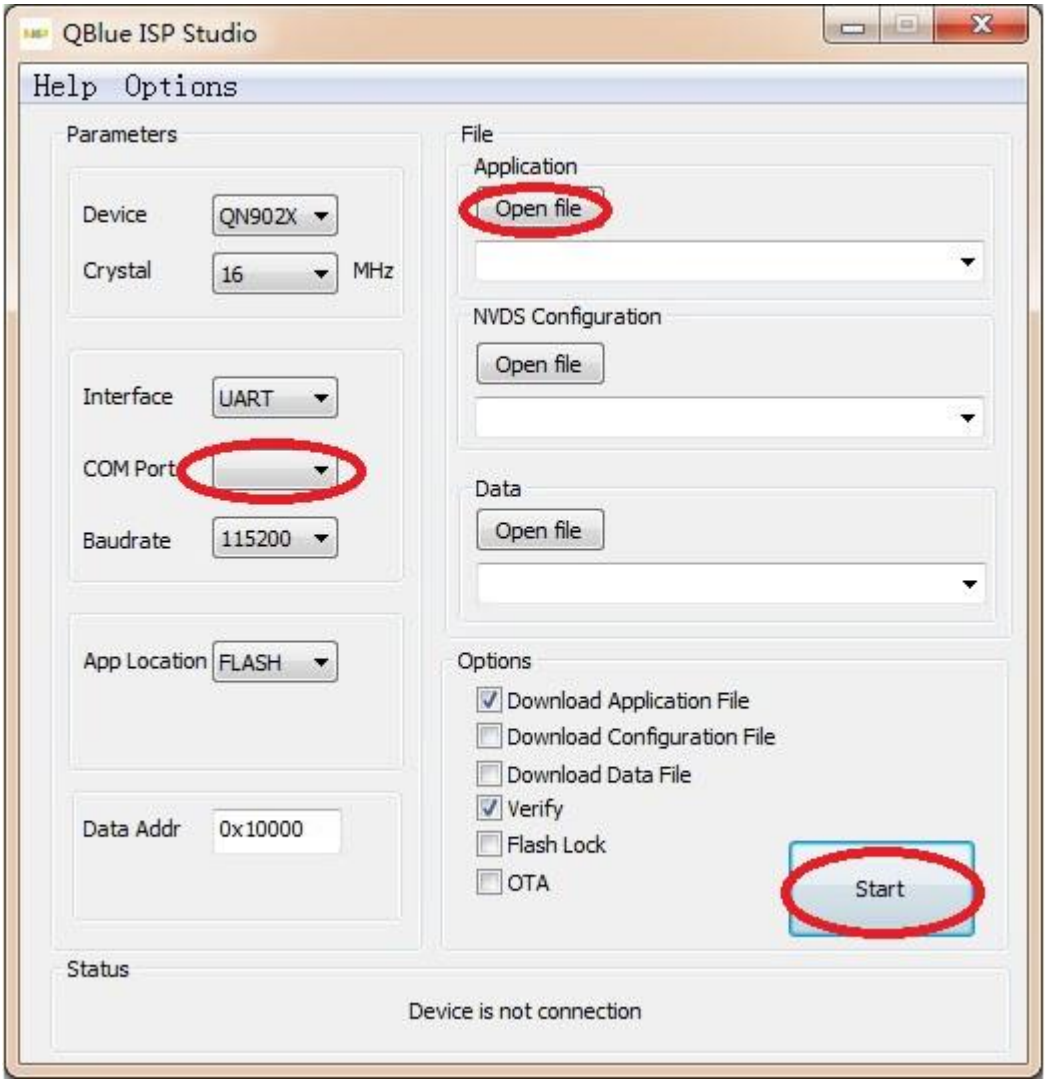

Figure 8 ISP Download bin file

6.2 Upgrade through OTA

1. If you need to encrypt the firmware, you must first run the qotapack.exe located in QBlue-x.x.x\Tools\qotapack.
   - Usage: qotapack [OPTION]…

Options:

-v, --version: Mandatory; Firmware version (Hex Format, 2bytes).

-e, --encrypt: Optional; Enable to encrypt the original firmware.

-tp, --type: Optional.

0: Original file is firmware.

1: Original file is data.

2 : Two original files are code and data

-k, --key: If encryption is enabled, it is mandatory, otherwise it is ignored.

AES 128 key (Hex Format, 16bytes).

-f, --from: Mandatory; Original firmware file. I

-f2, --from2: Optional; Original data file.

-t, --to: Mandatory; Encrypted firmware file.

-h, --help

- Examples:

  Examples 1: Firmware only encryption:
  ‘ qotapack --version=1234 --encrypt --key=11223344556677889900AABBCCDDEEFF --from=origianl.bin --to=encrypted.bin’

  Example 2. Data only encryption:
  ‘ qotapack --version=0x2001 --encrypt --type=1 --key=11223344556677889900AABBCCDDEEFF --from=data.bin --to=encrypted.bin’

  Example 3. Firmware and data encryption:
  ‘ qotapack --version=0x2001 --encrypt --type=2 --key=11223344556677889900AABBCCDDEEFF --from=firmware.bin -f2=data.bin --to=encrypted.bin’

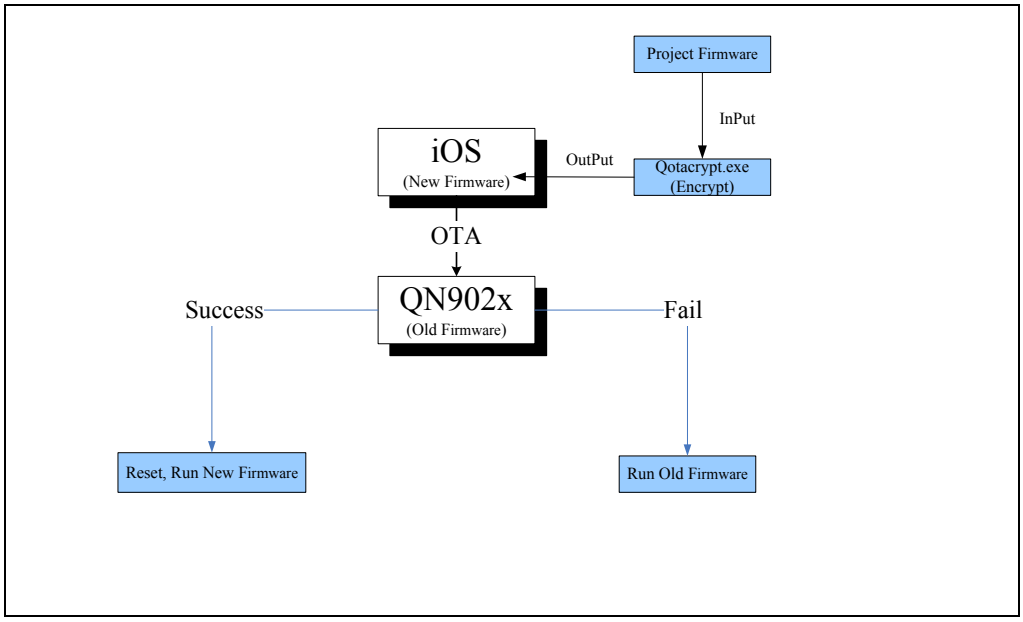

Figure 9 iOS Download and Encrypt firmware file

2. Install the client app in an Android or IOS device.
3. Place the firmware 2 in right location in Android/IOS devices

There is a demo bin file (firmware 2) named as 'ota_pack.bin' in the path '\QBlue\QN9020\QBlue-X.X.X\BinFiles'. For Android, the file should be put into the folder 'sd card/NXP_BLE' on the target phone.

For IOS, the file should be put into the folder "Documents" which is located in the example iOS application by some tools, such as iTools.

4. Start the app in Android/IOS and initiate the upgrade procedure.

Then the new firmware (firmware 2) in Android/IOS phone will be downloaded and upgraded to BLE device.

Note:

If your new version of the app (Firmware 2) doesn't have OTA function, you can only upgrade once.

7. References

Included with QBlue-X.X.X Release. The QBlue-X.X.X software has been installed to the default path 'C:\ QBlue\QN9020\QBlue-X.X.X':

[1] C:\QBlue\QN9020\QBlue-X.X.X\ Documents\QBlue ISP Studio Manual v1.0.pdf

[2] C:\QBlue\QN9020\QBlue-X.X.X\ Documents\QN9020 API Programming Guide v1.0.pdf

[3] OTA Profile Guide

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. List of figures