OTA Integration Guide for iOS

Version 0.1

Table of Contents

1. Introduction 1
2. Overview 1
3. Flowchart 2
4. API and Delegate Description 3
    4.1 otaStart() 3
    4.2 didOtaAppProgress() 3
    4.3 didOtaAppResult() 4
    4.4 Integration Note 4
5. Example code 5
Release History 6

1. Introduction

The OTA (Over the Air) profile is used to upgrade the firmware of QN902x over BLE. The libQBlueOta library acts as OTA client role, which is used by application to upgrade the firmware of the OTA server.

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 was broken during the upgrade progress, we just re-connect and continue download firmware while we don't have to start from the beginning.

Note: In order to upgrade succeed, please guarantee that the firmware run-time size (includes Code, RO-data, ZI-data, RW-data) must be less than 50K bytes.

2. Overview

The OTA 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 is shown in Figure 1:

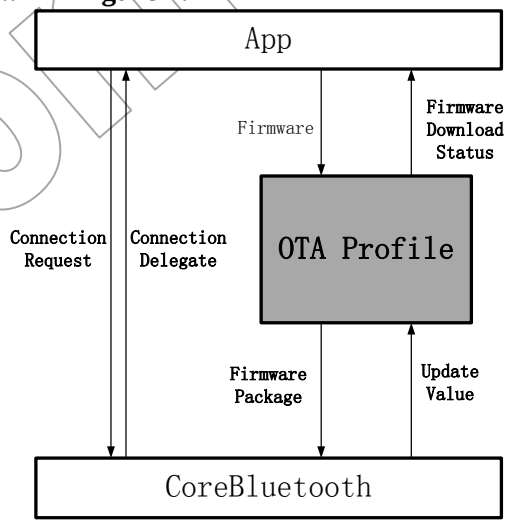

Figure 1 OTA Diagram

3. Flowchart

The OTA general flowchart is the following:

- Scan BLE peripherals around.
- 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 that whether the OTA is Success or failure.

OTA flowchart is shown in Figure 2:

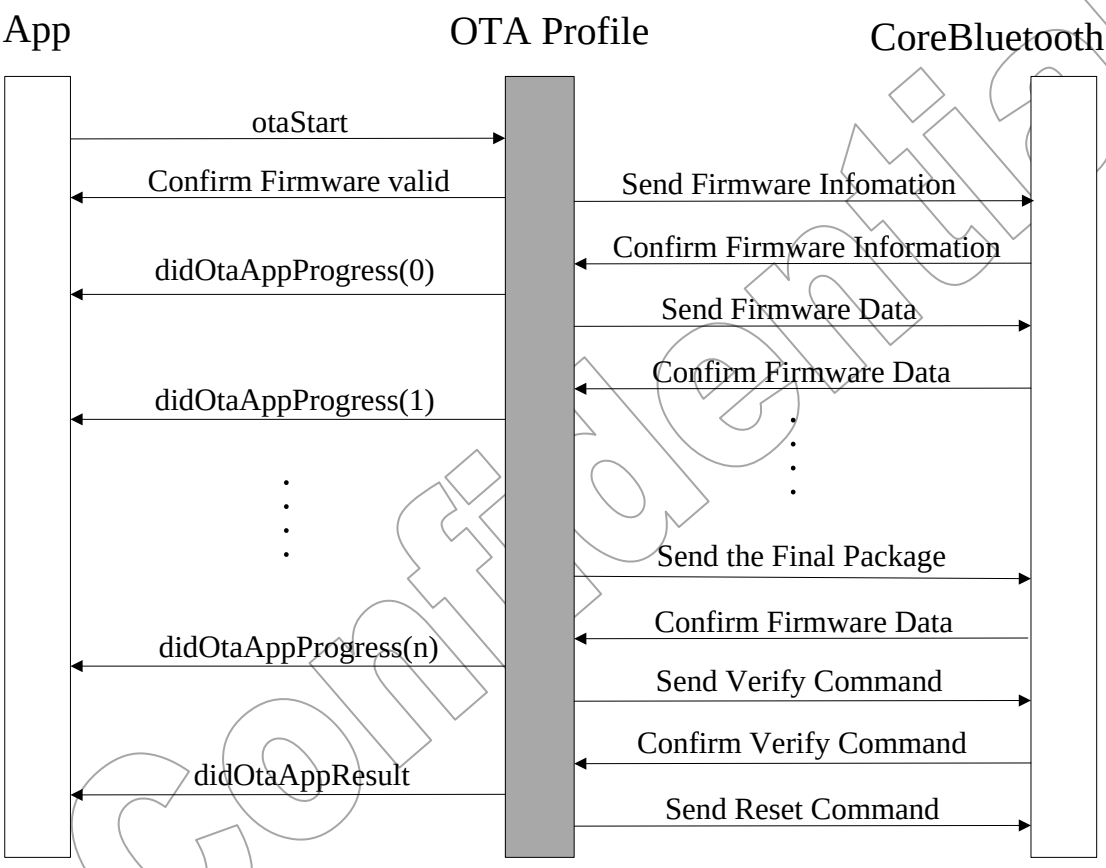

Figure 2 OTA flowchart

4. API and Delegate Description

These functions consist of one API functions and two delegate functions. API functions implement to register user's UUID and start OTA. The delegate functions implement update the status of the current transmitting package and the final OTA update result.

4.1 otaStart()

Prototype:

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

Parameters:

In	aPeripheral	The target peripheral with OTA profile
In	firmwareAddr	firmware 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.

Description: The function is used by the application (which handles the OTA application) to upgrade a firmware file through the OTA client role. After a peripheral is connected, then OTA service and characteristics are discovered. Once the API function is called, the OTA state machine starts work till end. During the upgrade process, if the connection is broken and the user resumes the connection. This API could be called again with the parameter *fResume* as TRUE. The upgrade process will resume from where the connection was broken.

4.2 didOtaAppProgress()

Prototype:

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

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

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

4.3 didOtaAppResult()

Prototype:
-(void) didOtaAppResult : (enum otaResult) otaResult;

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.

Description: The function can be used by OTA application to refresh UI when the OTA end. For example, if it is failure, user can load firmware and start OTA once again, or check your hardware, or re-connect and re-start.

4.4 Integration Note

4.4.1 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 quintic profile delegate
      [bleUpdateForOtaDelegate bleDidUpdateCharForOtaService : aPeripheral
                                                  withService : service
                                                        error : error];

      /// user code
      ……

}
```


4.4.2 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. Example code

There is an example iOS project in QBlue SDK, and it can be found in "\Quintic Corporation\QBlue-X.X.X\Projects\iOS\Ota\". It shows how to use the libQBlueOta library to implement firmware upgrade.

There is a demo bin file named as "ota_pack.bin" in the path "\Quintic Corporation\QBlue-X.X.X\BinFiles". The file should be put into the folder "Documents" which is located in the example iOS application by some tools, such as iTools.

Release History

REVISION	CHANGE DESCRIPTION	DATE
0.1	Initial release	2014-05-19

