QPP Programming Guide

Rev. <1.2> — 4 April 2018 Application note

Document information

Info	Content
Keywords	QPP Server, QPP client in Android, QPP client in IOS
Abstract	This document demonstrates with example about how to create application working as QPP server in BLE peripherals device and application as QPP client role in BLE central device.

Revision history

Rev	Date	Description
0.1	20140519	Initial release
1.0	20150330	Merged programing in BLE, iOS client, Android client in one document; Migrated to NXP template.
1.1	20150925	Updated some description
1.2	20180404	Updated some description

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

The QPP (Proprietary Profile) is used to transfer the raw data between BLE devices. This document demonstrates with example about how to create application working as QPP server in BLE peripherals device and application as QPP client role in BLE central device.

2. QPP Server

2.1 Project Example

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

2.2 Software Description

The QPP application is implemented in the following files:

- app_qpps.c: Application QPPS API
- app_qpps_task.c: Task handling functions
- qpp.lib and qpps_task.h and qpp_common.h: QPP Profile

2.2.1 User Configuration

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

- #define **CFG_PRF_QPPS**
- #define **CFG_TASK_QPPS TASK_PRF8** (Mandatory)
- #define **QPPS_NOTIFY_NUM 7**(Max : 7 , Min : 0)

2.2.2 Initialization

The initialization of the application occurs in two phases: Firstly, the **qpps_init()** function is called by the profiles register function(**prf_init_reg(prf_init)**). This function register QPPS task into kernel. Secondly, the **app_qpps_create_db(uint8_t char_num)** function is called by the **app_create_server_service_DB()** function. This function used to create server service database, the application can define the number of Characteristics used to send data to a client through notify.
NOTE: char_num: Max=7 Min = 0. If char_num increases, transmission speed will be faster, but more and more space will be occupied.

2.2.3 Data Processing

The application has three data processing functions, app_qpps_data_send(), app_qpps_data_send_cfm_handler() and app_qpps_data_ind_handler(). The app_qpps_data_send() function is used by the application to send raw data. The app_qpps_data_send_cfm_handler() function is used to report to the application a confirmation. The app_qpps_data_ind_handler() function is used to handle the data sent form peer device.

The diagrams below shows the relationships between APP and Profile:

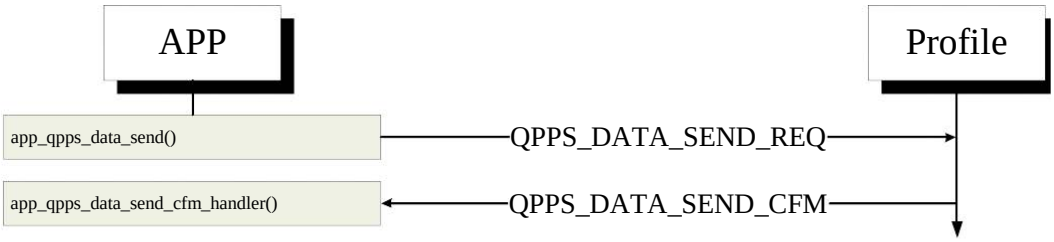

Figure 1 Data Sending

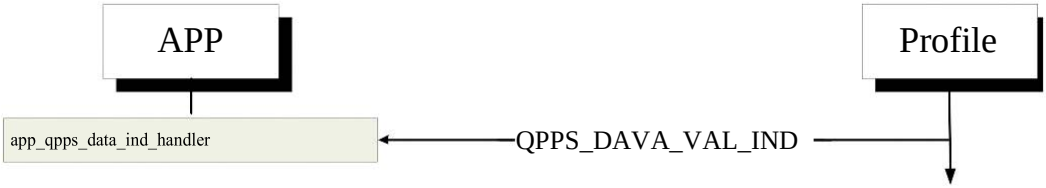

Figure 2 Data Receiving

2.3 API and Handler

2.3.1 qpps_init()

Prototype:

void qpps_init(void);

Description:

This function performs all the initializations of the QPPS module.

2.3.2 qpps_set_service_uuid()

Prototype:

void qpps_set_service_uuid(uint8_t param[ATT_UUID_128_LEN]);

Parameters:

in	param	QPPS's UUID

Description:

This function should be called before adding QPP service into the database.

2.3.3 app_qpps_create_db()

Prototype:

void app_qpps_create_db (uint8_t char_num);

Parameters:

in	char_num	The number of Characteristic used to send data

Response:
QPPS_CREATE_DB_CFM

Description:
This function shall be used to add an instance of the Proprietary Profile service into the database. This should be done during the initialization phase of the device.

Note:
Application can define the number of Characteristic used to send data to client through notify.

2.3.4 app_qpps_enable_req()

Prototype:
void app_qpps_enable_req (uint16_t conhdl, uint8_t sec_lvl, uint8_t con_type, uint16_t ntf_en).

Parameters:

in	conhdl	Connection handle
in	sec_lvl	Security level required for protection of HRS attributes: Service Hide and Disable are not permitted. Possible values are: PERM_RIGHT_ENABLE PERM_RIGHT_UNAUTH PERM_RIGHT_AUTH
in	con_type	Connection type: configuration(0) or discovery(1)
in	ntf_en	Notification configuration

Response:
None

Description:
This function is used for enabling the Server role of the Proprietary service.

2.3.5 app_qpps_data_send()

Prototype:
void app_qpps_data_send (uint16_t conhdl, uint8_t index, uint8_t length, uint8_t * data).

Parameters:

in	conhdl	Connection handle
in	index	Index of Characteristic to be sent
in	length	Length of data to be sent
in	data	Pointer to data to be sent

Response:
QPPS_DATA_SEND_CFM

Description:
This function is used by the application to send a raw data.

2.3.6 app_qpps_create_db_cfm_handler ()

Prototype:
int app_qpps_create_db_cfm_handler (ke_msg_id_t const msgid, struct qpps_create_db_cfm * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_CREATE_DB_CFM
in	param	struct qpps_create_db_cfm
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:
As it is a message handler, the related handling result for the message will be saved in related 'struct qpps_create_db_cfm * param' depending on the message was handled or not.

Description:
This handler will be triggered after a database creation. It contains status of database creation.

2.3.7 app_qpps_disable_ind_handler ()

Prototype:
int app_qpps_disable_ind_handler (ke_msg_id_t const msgid, struct qpps_disable_ind * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_DISABLE_IND
in	param	Pointer to the struct qpps_disable_ind
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:
As it is a message handler, the related handling result for the message will be saved in related 'struct qpps_disable_ind * param' depending on whether the message was handled or not.

Description:

This handler is used to inform the Application of a correct disable. The configuration that the client has set in ntf_en field must be conserved for bonded devices.

2.3.8 app_qpps_error_ind_handler ()

Prototype:

int app_qpps_error_ind_handler (ke_msg_id_t const msgid, struct qpps_error_ind * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_ERROR_IND
in	param	Pointer to the struct qpps_error_ind
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:

As a message handler, the result will be saved in the related 'struct qpps_error_ind * param' depending on whether the message was handled or not.

Description:

This handler is used to inform the Application of an occurred error.

2.3.9 app_qpps_data_send_cfm_handler ()

Prototype:

int app_qpps_data_send_cfm_handler (ke_msg_id_t const msgid, struct qpps_data_send_cfm * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_DATA_SEND_CFM
in	param	Pointer to the struct qpps_data_send_cfm
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:

As a message handler, the result will be saved in the related 'struct qpps_error_ind * param' depending on whether the message was handled or not.

Description:

This handler is used to report to the application a confirmation or error status of a notification request being sent by application.

2.3.10 app_qpps_cfg_indntf_ind_handler ()

Prototype:

int app_qpps_cfg_indntf_ind_handler (ke_msg_id_t const msgid, struct qpps_cfg_indntf_ind * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_CFG_INDNTF_IND
in	param	Pointer to the struct qpps_cfg_indntf_ind
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:

As it is a message handler, the related handling result for the message will be saved in related 'struct qpps_cfg_indntf_ind * param' depending on the message was handled or not.

Description:

This handler is used to inform application that peer device has changed notification configuration.

2.3.11 app_qpps_data_ind_handler ()

Prototype:

int app_qpps_data_ind_handler (ke_msg_id_t const msgid, struct qpps_data_val_ind * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)

Parameters:

in	msgid	QPPS_DAVA_VAL_IND
in	param	Pointer to the struct qpps_data_val_ind
in	dest_id	TASK_APP
in	src_id	TASK_QPPS

Returns:

As it is a message handler, the related handling result for the message will be saved in related 'struct qpps_data_val_ind * param' depending on the message was handled or not.

Description:

This handler is used to handle the data sent form peer device.

3. QPP Client Overview

The QPP (Proprietary Profile) is used to transfer the raw data between BLE devices.
The libQBlueQPP library acts as QPP client role, which is used by application to transfer and receive the raw data between BLE devices.

3.1 Features

Transmit free raw data between BLE devices. Single free raw data package maximum length is 20bytes, minimal is 1byte.

3.2 Overview

The QPP client diagram consists of three parts:

App Layer:

- Send connection requests to BluetoothGatt, and configure API layer.
- Send data to API layer.
- Receive data from API layer.

API Layer:

- Receive data from App layer and deliver the data received to BluetoothGatt.
- Receive data from BluetoothGatt and deliver the data received to App layer.

BluetoothGatt Layer:

- Receive request from API layer.
- Update value to API layer.

The QPP client diagram for Android is shown in Figure 1

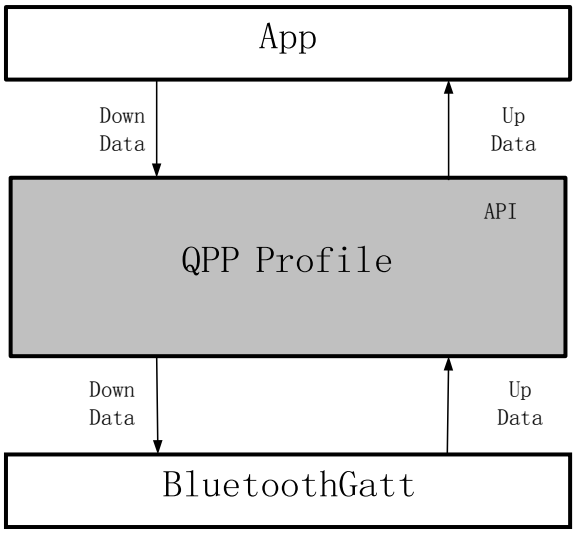

Figure 3 QPP Client Diagram for Android

The QPP client diagram for iOS is shown in

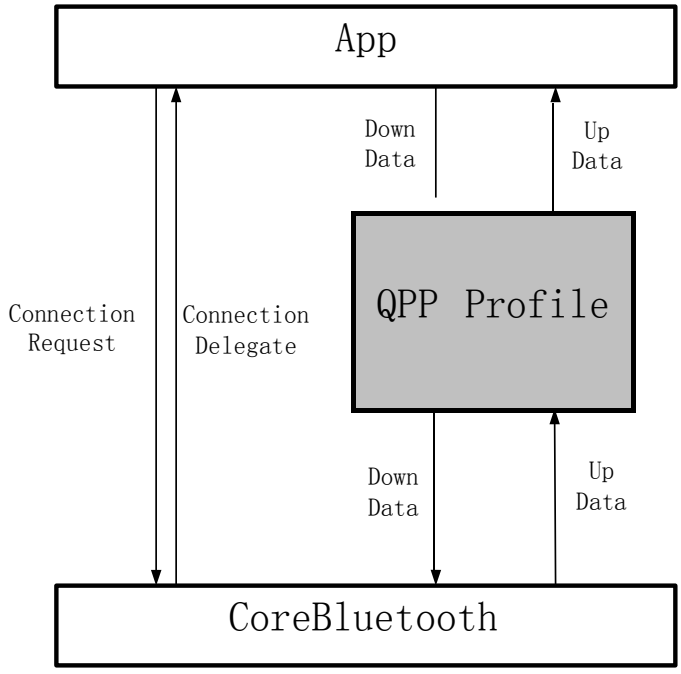

Figure 4 QPP Client Diagram for iOS

4. QPP Client Integration-Android

4.1 Flowchart

The QPP client general flowchart is the following:

- Scan BLE devices around.

- Establish a connection with the device which is built-in QPP profile server.
- Discover services and characteristics.
- Register user's special UUIDs (including QPP service UUID and write characteristic UUID), here you'd call the method: qppEnable.
- User receives data in the onQppReceiveData function, or sends data by the QppSendData function.

QPP TX flowchart is shown in Figure 5:

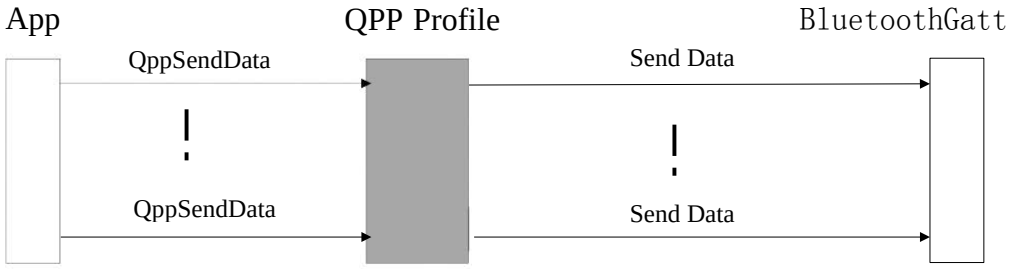

Figure 5 QPP Client TX flowchart

QPP RX flowchart is shown in Figure 6:

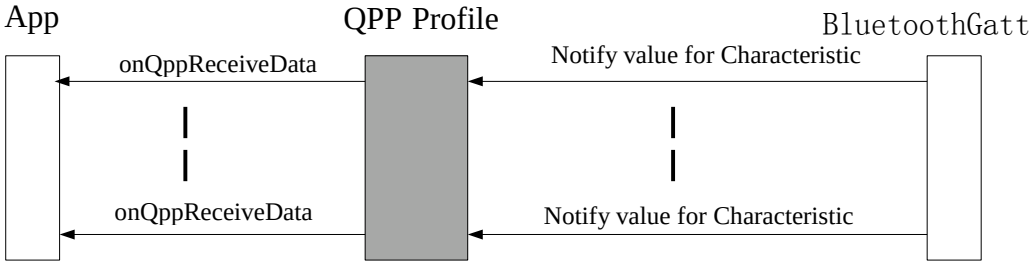

Figure 6 QPP Client RX flowcharts

1.1 API and Callback Description

There are one public class *QppApi* and one interface *iQppCallback* in the libQblueQpp library.
The class *QppApi* defines APIs. The interface *iQppCallback* declares callbacks. There are five functions relevant: three API functions and two callback functions. These API functions are responsible to enable register service's UUIDs, transfer data. These callback functions are used to receive data, get QPP service status.

4.1.1 Class QppApi

4.1.1.1 General Definition

```
public class QppApi {
    public static boolean qppEnable(BluetoothGatt bluetoothGatt, String
  qppServiceUUID, String writeCharUUID);
    public static boolean qppSendData(BluetoothGatt bluetoothGatt, byte[]
  qppData);
    public static boolean setQppNextNotify(BluetoothGatt bluetoothGatt, boolean
   EnableNotifyChara);
    public static void  updateValueForNotification(BluetoothGatt bluetoothGatt,
    BluetoothGattCharacteristic characteristic);
    public static void setCallback(iQppCallback mCb);
};
```

4.1.1.2 API Description

public static boolean qppEnable()

Function public static boolean qppEnable(BluetoothGatt bluetoothGatt, String qppServiceUUID, String writeCharUUID);

Brief Register customer's UUIDs, in order to support customer's devices using customized QPP UUIDs.

Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler
In	qppServiceUUID	UUID for QPP service in string
In	writeCharUUID	UUID for write Characteristic in string

Returns:

True The service is found and bluetoothGatt is not null.
False The service is not found or bluetoothGatt is null.

Note:

The qppServiceUUID must match the QPP UUID on the device side.

public static boolean qppSendData()

Function public static boolean qppSendData(BluetoothGatt bluetoothGatt, byte[] qppData);

Brief Send raw data to QPP Profile.

Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler
In	qppData	Data to send, the length should not be larger than 20bytes

Returns:

True Argument is valid and sends data is successful.
False Argument is invalid or sends data is failed.

public static boolean setQppNextNotify ()

Function public static boolean setQppNextNotify(BluetoothGatt bluetoothGatt,

boolean EnableNotifyChara);
Brief Enable characteristics notification.
Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler
In	EnableNotifyChara	'true' to enable and 'false' to disable

Returns:
True set characteristics is successful.
False set characteristics is failed.

public static boolean updateValueForNotification ()

Function public static void updateValueForNotification(BluetoothGatt bluetoothGatt, BluetoothGattCharacteristic characteristic);
Brief Notify libQblueQpp that data have been received.
Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler
In	characteristic	Notify characteristic

Returns:
None.
Note:
This function should be invoked in BluetoothGattCallback. onCharacteristicChanged.

public void boolean setCallback ()

Function public static void setCallback(iQppCallback mCb);
Brief Set callback function handler.
Parameters:

In	mCb	iQppCallback object

Returns:
None.

4.1.2 Interface iQppCallback

4.1.2.1 General Definition

```
public interface iQppCallback {
    void onQppReceiveData(BluetoothGatt bluetoothGatt, String
qppUUIDForNotifyChar, byte[] qppData);
}
```

4.1.2.2 API Description

void onQppReceiveData()

Function void onQppReceiveData(BluetoothGatt bluetoothGatt, String qppUUIDForNotifyChar, byte[] qppData);
Brief Process the data that received from QPP Profile.
Parameters:

In	bluetoothGatt	Android BluetoothGatt client handler

In	qppUUIDForNotifyChar	UUID for notify characteristics.
Out	qppData	The received data from the notify characteristics.

Returns:
None.

4.2 Integration Note

4.2.1 Initialize

4.2.1.1 Add 'QppApi.qppEnable' method

The method is used by the application to register user's UUIDs in order to support customer's devices using customized QPP UUIDs. The qppServiceUUID must match the QPP UUID on the device side. Then profile discovery the service, characteristic from bluetoothGatt and enable notification characteristics to bluetoothGatt. The parameter bluetoothGatt is a connected BluetoothGatt.

Add this method in following function:

```
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback(){
{
    … …
    public void onServicesDiscovered(BluetoothGatt bluetoothGatt, int status) {
        if(QppApi.qppEnable(bluetoothGatt, uuidQppService, uuidQppCharWrite))
            isInitialize = true;
    }
       … …
 }
```

4.2.2 Rx Data

4.2.2.1 Add 'QppApi.setQppNotify()' method

This method is to enable the QPP notification characteristics.
Add this method in following function:

```
public void onDescriptorWrite(BluetoothGatt bluetoothGatt, BluetoothGattDescriptor
                                        descriptor, int status)
{
  QppApi.setQppNextNotify(bluetoothGatt, true);
    /// user code
}
```

4.2.2.2 Add 'QppApi.updateValueForNotification' method

This method is to update value for notification characteristic.
Add this method in following function:

```
public void onCharacteristicChanged(BluetoothGatt bluetoothGatt,
BluetoothGattCharacteristic characteristic)
{
      QppApi.updateValueForNotification(bluetoothGatt, characteristic);
      /// user code
}
```

4.2.2.3 Receive data

Refer **to chapter 4.2.2** onQppReceiveData().

4.2.3 Tx Data

Call QppApi .qppSendData() to write data

```
public void onCharacteristicWrite(BluetoothGatt bluetoothGatt,
                                  BluetoothGattCharacteristic
                                  characteristic,int status)
{
   handlersend.postDelayed(runnableSend,20);
}

private Handler handlersend = new Handler( );

final Runnable runnableSend = new Runnable( )
{
   public void run ( )
     {
      QppApi.qppSendData(bluetoothGatt, qppDataSend);
   }
};
```

4.3 Example code

There is one example named as *'QPP_Android_xxx.zip'* in Collabnet which shows how to use the lib*'libQBlueQPP.jar'*
Qpp\libQBlueQpp\bin to transfer raw data between QN902x device and QPP client.

5. QPP Client Integration-iOS

5.1 Flowchart

The QPP general flowchart is the following:

- Register user's special UUIDs (including QPP service UUID and write characteristic UUID), here you'd call the method: *qppRegUUIDs*.
- Scan BLE peripherals around.
- Establish a connection with the device which is built-in QPP profile server.
- Discover services and characteristics.
- User receives data in the *didQppReceiveData* delegate function, or sends data by the *qppSendData* function.

QPP TX flowchart is shown in Figure 7:

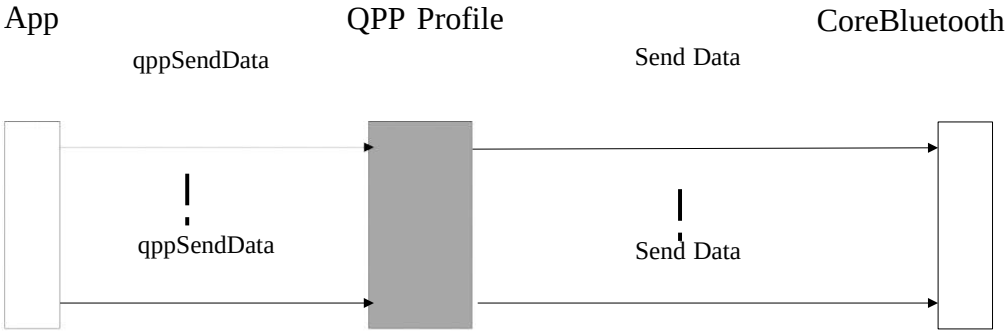

Figure 7 QPP TX flowchart

QPP RX flowchart is shown in Figure 8:

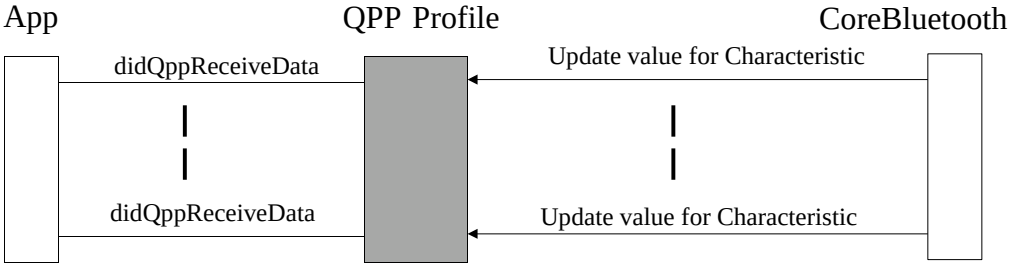

Figure 8 QPP RX flowchart

5.2 API and Delegate Description

These functions consist of two API functions and one delegate function. API functions implement to register user's UUIDs and to transfer data, delegate function used to receive data.

5.2.1 qppRegUUIDs()

Prototype:

(void)qppRegUUIDs : (NSString *)qppServiceUUID
withWrChar : (NSString *)writeCharUUID

Parameters:

in	qppServiceUUID	UUID for QPP service in string
in	writeCharUUID	UUID for write Characteristic in string

Returns:

None.

Description: The method is used by the application to register user's UUIDs in order to support customer's devices using customized QPP UUIDs. The *qppServiceUUID* must match the QPP UUID on the device side. The method is called before discovery procedure.

5.2.2 qppSendData()

Prototype:

(void)qppSendData : (CBPeripheral *)aPeripheral
withData : (NSData*)qppData;

Parameters:

in	aPeripheral	The peripheral must be built-in QPP profile server
in	qppData	The raw data

Returns:

None.

Description: The function is used by application to send raw data to QPP Profile.

5.2.3 didQppReceiveData()

Prototype:

(void)didQppReceiveData : (CBPeripheral *)aPeripheral
withCharUUID : (CBUUID *)qppUUIDForNotifyChar
withData : (NSData *)qppData;

Parameters:

Out	aPeripheral	The data received is from the peripheral.
Out	qppUUIDForNotifyChar	The UUID for notify characteristics.
Out	qppData	The data received is from the notify characteristics.

Returns:

None.

Description: The function is used by application to process the data received from QPP Profile.

5.3 Integration Note

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

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

```
                                withService : aService
                                    error : error];

    /// user code
    ……
}
```

b) Please insert the "***bleDidUpdateValueForQppChar***" 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)
    {
        [bleUpdateForQppDelegate
                bleDidUpdateValueForQppChar : (CBPeripheral*)aPeripheral
                    withService : (CBService *)aService
                      withChar : (CBCharacteristic *)characteristic
                         error : (NSError *)error];
        /// user code
        ……
    }
}
```

5.4 Example code

There is an example iOS project named 'QPP_IOS_xxx.zip' in Collabnet. It shows how to use the libQBlueQPP library to implement transfer raw data between QN902x device and QppDemo.

6. References

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

[1] C:\QBlue\QN9020\QBlue-X.X.X\ Documents\
QN9020 Device Database for IDE User Manual v1.0.pdf

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

[3] QN9020 Software Developer's Guide v1.5.pdf

7. Legal information

7.1 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.

7.2 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.

7.3 Trademarks

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

8. List of figures