

QBluetoothZero - A Qt bluetooth library

QBluetoothZero is a **3rd party** bluetooth library written on Qt supported on Symbian and Windows. Its architecture is based on [S60 Platform: Bluetooth API Developer's Guide](#).



Warning: The project has been renamed from QBluetooth to QBluetoothZero to avoid conflicts with Bluetooth module of Qt Mobility. Users that have already used this library are advised to have a look at the following wiki page which states the changes that need to be made to comply with the new version of the library: [From QBluetooth to QBluetoothZero](#)

The idea is to create a library implementation using Qt (a DLL) of the S60 bluetooth API easy to use by anyone with basic knowledge of Qt.

So the architecture was extracted from S60 bluetooth API, and also source code was used from the following examples:

- [S60 Platform Bluetooth OBEX Example](#)
- [S60 Platform Bluetooth Point to Multipoint Example](#)
- [S60 bluetooth Serial Port Chat](#)

and the basic guide for combining all these was [Using Qt and Symbian C++ Together](#)

NOTE: for more detailed description of the the library's components please advise the documentation. Documentation can be found in this site [QBluetoothZero Documentation](#). To **download** source code visit [project page](#).

An application using QBluetoothZero can be found here [QuteMessenger - A Qt application using QBluetoothZero](#)

Some code examples from the Nokia Wiki:

- [Discovering nearby Bluetooth services with the QBluetooth library](#)
- [Discovering Bluetooth devices with the QBluetooth library](#)

Basic Information Container Classes

- *QbtAddress* : that represents the device's bluetooth address
- *QbtService* : all the necessary information about a bluetooth service
- *QbtDevice* : all the information needed about any remote bluetooth device
- *QbtLocalDevice* : implemented only with static functions through which user can access information of the local device

Operational Classes

- *QbtDeviceDiscoverer*: the mechanism of the device discovery
- *QbtSingleDeviceSelectorUI*: a simple widget to select a bluetooth device. Device discovery is initiated when widget is shown. Created for convenience if a user simply wants to select a discovered device to proceed to other operations
- *QbtServiceDiscoverer*: provides the mechanism to inquire a given device for the bluetooth services it supports
- *QbtServiceAdvertiser*: class responsible to advertise a given bluetooth service
- *QbtObjectExchangeClient* : OBEX client, connects to a remote OBEX server to send/retrieve data and files
- *QbtObjectExchangeServer* : OBEX server, waits for clients to connect and ready to serve any OBEX request
- *QbtSerialPortClient* : Serial Port (SP) client, connect to SP server to send/receive raw data through serial port
- *QbtSerialPortServer* : Serial Port server, ready to accept client connections to also send/receive raw data

Building the library

1. . You will need a standalone SDK, like the S60 5th Edition or 3rd Edition FP2, for example. Download them from [Nokia Developer](#).
2. . The library requires the BluetoothEngineAPI. This API is available as a SDK plugin. Please see [SDK API Plug-in](#) to see how to get it.
3. . Download and install Qt.

Use QBluetoothZero.pro file to open the project. Next step is create the DLL.

If compiled for Symbian and Carbide is used as IDE then we Freeze Exports (Project->Freeze Exports) and it is ready. If compiled for Windows (in case of Visual Studio 2008 as IDE) just compile and the .dll and .lib files will be generated.

It's also possible to build the library with Nokia Qt SDK 1.0. However, the Symbian SDK that comes with Nokia Qt SDK 1.0 does not have all the libraries required to build the library (eg. BluetoothEngineAPI). The options are 1) copy the missing files from the 5th Edition SDK, or 2) choose the standalone SDKs as the Qt Version when configuring a project (Symbian device target).

Possible build errors

This section provides answers to some questions that might arise when building the library.

Qt Creator doesn't find the Symbian SDK header files

Add this to the QBluetoothZero .pro file, just before the HEADERS declaration at the beginning of the file.

```
# fix for Qt Creator to find the symbian headers
INCLUDEPATH += $$EPOCROOT\epoc32\include
```

Compiler error in <obexbase.h>, line 205: extra qualification 'COBex::' on member '!CancelObexConnection'.
Edit the obexbase.h file and remove the COBex:: qualifier.

Qt Creator complains about missing libraries

The solution found so far for S60 5th Edition SDK and Nokia Qt SDK 1.0 is to install the BluetoothEngineAPI files from the S60 3rd Edition FP2 plugin pack. For the S60 3rd Edition FP1 SDK, please search for the plugin pack specific for this SDK.

If Qt Creator complains about `{{-lbttextnotifiers}}` or `{{-lirobex}}`, when using as target Qt devices (Nokia Qt SDK 1.0): The solution so far is to either use another SDK (as the 5th Edition SDK), or to copy the missing files from the 5th Edition SDK). The files are:

```
btextnotifiers.dso
btextnotifiers.lib
btextnotifiers{000a0000}.dso
btextnotifiers{000a0000}.lib

irobex.dso
irobex.lib
irobex{000a0000}.dso
irobex{000a0000}.lib
```

These files are located in `\epoc32\release\armv5\lib` of the SDK installation directory.

Qt Creator doesn't generate the .sis file for the library

No known solution so far. Even if having configured the build target to sign the file with a developer certificate, Qt Creator (Nokia Qt SDK 1.0) does not generate the sis file. It seems to be a bug in Qt Creator.

To make Qt Creator generate the .pkg file, add this to the QBluetoothZero.pro file at the end of the file:

```
addFiles.sources = QBluetoothZero.dll
addFiles.path = !:\sys\bin
DEPLOYMENT += addFiles
```

Compiler errors when modifying the public API (any public method of QBluetoothZero classes), with Qt Creator

When modifying the public API in Symbian DLLs, it's necessary to freeze the API. Currently, Qt Creator does not do this, and there's no way to instruct it to make this step. This is a known Qt Creator bug (<https://bugreports.qt-project.org/browse/QTCREATORBUG-772>).

To freeze the API it's necessary to run it from the command line. Create a .bat file and place it in the same directory where the QBluetoothZero .pro file resides. Write the text below:

```
set EPOCROOT=\Dev\Symbian\S60_3rd_FP1\
del eabi\*.def
abld freeze gcce
```

Set the **EPOCROOT** variable to the root of the Symbian SDK that is being used by the project (without the drive letter as in the example). For Nokia Qt SDK 1.0, this is `sdk instalation folder\Symbian\SDK`. Then, proceed as this:

1. . Clean the project with Qt Creator
2. . Build the project with Qt Creator
3. . Run the .bat file
4. . Build the project again with Qt Creator

Including the library

Use the output of the previous build

The following lines must be inserted in the .pro file of the project that is going to use the QBluetoothZero library.

If compiling for Symbian, add in *symbian* scope:

```
LIBS += -lQBluetoothZero
symbian {
    INCLUDEPATH += /epoc32/include/QBluetoothZero

    TARGET.CAPABILITY = LocalServices \
        NetworkServices \
        ReadUserData \
        UserEnvironment \
        WriteUserData

    addFiles.sources = QBluetoothZero.dll
    addFiles.path = !:\sys\bin
    DEPLOYMENT += addFiles
}
```

If compiling for Windows, add in *win32* scope (supposing that the folder QBluetoothZero containing all the headers is in the same directory as the project that is going to use it):

```
LIBS += -lQBluetoothZero \
        -lQBluetoothZero\bin\release
INCLUDEPATH = QBluetoothZero
HEADERS += QBluetoothZero.h
```

The LIBS label sets the location of the file containing the .lib file, the INCLUDEPATH sets the location of the header file and HEADERS is used to specify the exact header to be included.

At any case, whenever one of the library classes is to be used, the developer must include <QBluetoothZero>.

Use the binaries

In **bin** folder you can find the library already compiled and ready to be used.

The binary is a Shared DLL Library. It is installed once and can be used from multiple client programs without conflicts.

It looks good BUT it has some catches.

- the binaries are *self signed* so only the basic Capabilities are supported: !LocalServices, !NetworkServices, !ReadUserData, !UserEnvironment, !WriteUserData, Location.

- The UID of the binary comes from the protected range. This means that it is only for testing and personal use. There is no guaranty that the UID will not conflict with programs from other developers.

- In case someone wants to use the binary for an application that will be placed in Ovi Store, the binary has to be rebuilt using a UID that the developer got a publisher.

- Step 1

Copy the contents of bin file ("epoc32" folder) in the "epoc32" folder of your Symbian SDK.

- Step 2

Put the following lines in your .pro file:

```
symbian{
    INCLUDEPATH += /epoc32/include/QBluetoothZero
    LIBS += -lQBluetoothZero

    customrules.pkg_prerules = \
        ";QBluetoothZero" \
        "@\"$(EPOCROOT)Epoc32/InstallToDevice/QBluetoothZero_selfsigned.sis\", (0xA003328D)\"\\
        \"\"
    DEPLOYMENT += customrules
}
```

In case you use Qt Creator, the following line:

```
"@\"$(EPOCROOT)Epoc32/InstallToDevice/QBluetoothZero_selfsigned.sis\", (0xA003328D)\"\\
```

must be changed into (just add a '\$' character):

```
"@\"$$$(EPOCROOT)Epoc32/InstallToDevice/QBluetoothZero_selfsigned.sis\", (0xA003328D)\"\\
```

- Step 3

Compile your application.

Code examples

Discovering devices in range

[QBluetoothDeviceDiscoverer](#) is used to inquire for bluetooth devices in range. - create new instance - call [startDiscovery\(\)](#) to initiate inquiry - call [getInquiredDevices\(\)](#) to retrieve the devices found so far or connect to SIGNAL [newDeviceFound \(QBluetoothDevice\)](#) to be reported each time a device is found

```
QBluetoothDeviceDiscoverer* deviceDiscoverer = new QBluetoothDeviceDiscoverer(this);
connect(deviceDiscoverer, SIGNAL(newDeviceFound (QBluetoothDevice)),
        this, SLOT(handlerFunction(QBluetoothDevice)));
deviceDiscoverer ->startDiscovery();
```

Discovering services on a device

[QBluetoothServiceDiscoverer](#) is used to inquire a remote device for the services it supports. After instantiation, user can call one of the 3 overloads of [startDiscovery\(\)](#) - [startDiscovery\(QBluetoothDevice*\)](#) : get all the device's supported services - [startDiscovery\(QBluetoothDevice*,QBluetoothConstants::!ServiceClass\)](#) : get all the device's supported services whose UUID equals to the given - [startDiscovery\(QBluetoothDevice*,QBluetoothConstants::!ServiceProtocol\)](#) : get all the device's supported services whose mechanism contains the service protocol given as argument

Use `getInquiredServices()` to get the services found so far or connect to the SIGNAL `newServiceFound(const QBluetoothDevice&, QBluetoothService)` to be reported each time a service is found.

Example: inquire a device (variable name `remoteDevice`) for all the Serial Port Protocols (SPP) it supports.

```
QBluetoothServiceDiscoverer* serviceDiscoverer = new QBluetoothServiceDiscoverer(this);
connect (serviceDiscoverer, SIGNAL(newServiceFound(const QBluetoothDevice&, const QBluetoothService&)),
        this, SLOT (handlerFunction(const QBluetoothDevice&, const QBluetoothService& )));

serviceDiscoverer->startDiscovery (remoteDevice, QBluetoothConstants::SerialPort);
```

Publish a service

`QBluetoothServiceAdvertiser` is responsible to advertise a given bluetooth service.

After the instantiation of the class, `startAdvertising(const QBluetoothService&)` can be called to start advertising the given bluetooth service. If successful then `advertisingStarted(const QBluetoothService&)` signal is emitted.

NOTE: Currently there is no implementation on the Windows platform so after the instantiation of this object, any call to any function will emit an `error(QBluetoothServiceAdvertiser::FeatureNotSupported)` signal.

```
QBluetoothService newService;
newService.setName(p_ptr->trService->getName());
newService.setClass(QBluetoothConstants::SerialPort);
newService.setPort(aChannel);
newService.addProtocol(QBluetoothConstants::L2CAP);
newService.addProtocol(QBluetoothConstants::RFCOMM);

QBluetoothServiceAdvertiser* advertiser = new QBluetoothServiceAdvertiser(this);
advertiser->startAdvertising(newService);
```

Send/receive a file

`QBluetoothObjectExchangeClient` is used to connect to a remote OBEX server and send or receive files or raw data.

Example connecting to a remote OBEX server. At successful connection user send a file to the server.

Header

```
#include <QBluetoothZero>

class FileSender
{
    Q_OBJECT
public:
    FileSender();
protected:
    void Setup();
private slots:
    void SendFile();
private:
    QBluetoothObjectExchangeClient* client;
}
```

Source

```
FileSender::FileSender()
{
    Setup();
}

void FileSender::Setup()
{
    client = new QBluetoothObjectExchangeClient(this);
    connect(client, SIGNAL(connectedToServer()), this, SLOT(SendFile()));
    client->connectToServer(remoteDevice, remoteService); // remoteDevice & remoteService are acquired from the previous examples
}

void FileSender::SendFile()
{
    client->putFile("C:\\picture.jpg");
}
```

OBEX server

`QBluetoothObjectExchangeServer` is used to create an OBEX server. After instantiation user can call `startServer(const QString&)` to start the server. The feedback from the server's operations is given through SIGNALS.

```
QBluetoothObjectExchangeServer* server = new QBluetoothObjectExchangeServer(this);
server->startServer("MyObexServer");
```

* NOTE: not implemented for Windows.

Connecting to a Serial Port Server

In this case we use `QBluetoothSerialPortClient`. Call `connect(const QBluetoothDeviceAddress&, const QBluetoothServiceUuid&)` to connect to a remote Serial Port (SPP) Server. If successful client can send data using `sendData(const QByteArray &)` function and reports received data through SIGNAL `dataReceived(const QByteArray)`.

Creating a Serial Port Server

`QBluetoothSerialPortServer` is used. Call `startServer(const QByteArray&)` to start server and set the name of the publishing service and then wait for a client to connect. When a client is connected, it is reported through `clientConnected(QBluetoothDeviceAddress)` SIGNAL. At this point server is able to send data through `sendData(const QByteArray)` and the data received are reported through `dataReceived(QByteArray)`.

- NOTE: not implemented for Windows.

Other useful information

On the Windows implementation side, the bluetooth functions are provided by the Bluesoleil SDK. So in order to use the Windows implementation user must download the Bluesoleil SDK and Bluesoleil Drivers 6.4 and above. I know that this restriction is a drawback but I had a couple of months of experience on that SDK so it was the only chance for me to make a Windows implementation at this moment...but it works.

- NOTE: if someone wants to use this implementation, consider to update the path of the folder containing the SDK in the `QBluetoothZero.pro` in the `win32` scope.

The Win Serial port functions are provided by Thierry Schneider through `Tserial_event`:

Copyright © 2001-2002 Thierry Schneider
thierry@tetraedre.com

Licensing

This project is licensed under the Apache License, Version 2.0.

A few code portions are used from other Nokia code examples thus in the root directory of the project there are these two files:stating the license of each.

- `License_OBEX_example.txt`
- `License_Point-to-Multipoint_Example.txt`

Future work

The project is further developed to its [project page](#).

Links

- Project's page
- The `QBluetoothZero` library documentation
- `QuteMessenger` - A Qt application using `QBluetoothZero`
- S60 Platform: Bluetooth API Developer's Guide
- S60 Platform Bluetooth OBEX Example
- S60 Platform Bluetooth Point to Multipoint Example
- S60 bluetooth Serial Port Chat
- Bluetooth Protocol
- Service Discovery in Bluetooth

favoritas37 00:29, 5 January 2012 (EET)

