This Guide Sets-up Communication Between Devices Using MQTT Protocol

Before we start

In this guide, we are going to cover how to setup and use [Eclipse's Paho Embedded MQTT library](https://github.com/eclipse/paho.mqtt.embedded-c) in your C++ project for BeagleBone Black/Green. We will not cover how to work with MQTT or how to use the library (although there are some notes at the end).

MQTT protocol is an extremely lightweight, flexible, and reliable publish/subscribe messaging transport. It was designed with IoT devices in mind so using MQTT with embedded systems is a perfect combination. We used it for our project, Facebook uses it for their Messenger service, and you should too.

Read more about MQTT here:
* http://mqtt.org/
* https://www.hivemq.com/blog/how-to-get-started-with-mqtt/
* http://mqtt.org/documentation
* Google

Advantages of MQTT:
* Fast. Imagine the speed of Facebook's Messenger push notifications. That's how fast it is.
* Reliable. You can have different levels of confirmation (QoS) to ensure your messages never get lost.
* Secure (if you use secure MQTT, not going to cover).
* Easy to use. You don't have to deal with IP addresses. Everything talks with each other using topics which are plain text! (However you do need to know broker's IP address in advance, more about this later.)
* You can have one-to-one, one-to-many, or many-to-one connections.

Setup

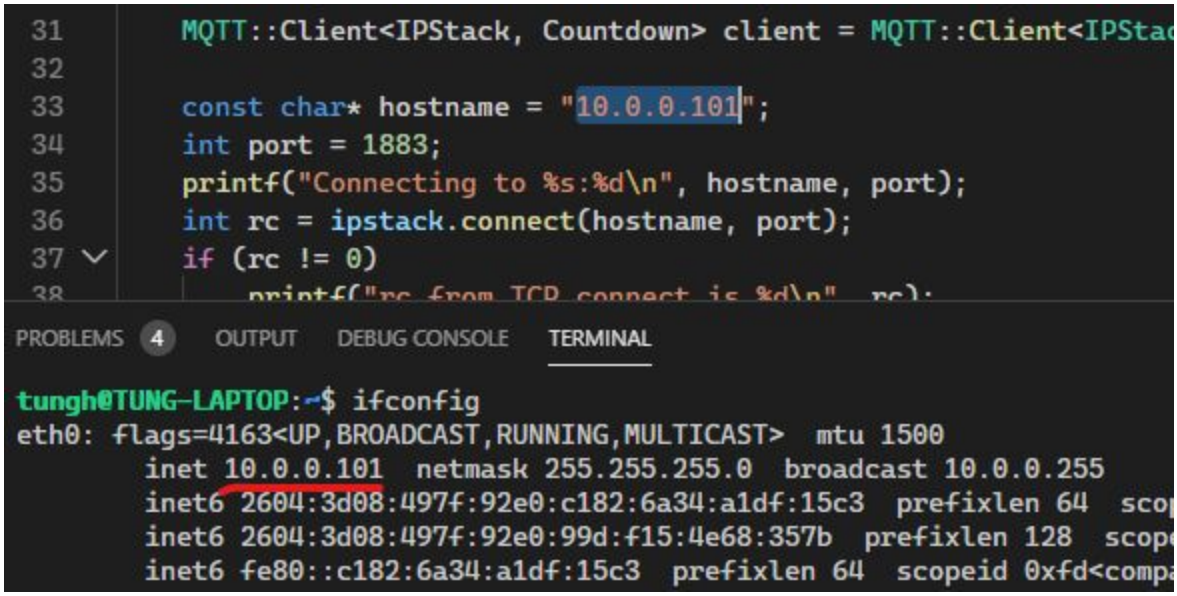
Download the library [here](https://github.com/eclipse/paho.mqtt.embedded-c/tree/develop). You don't need to clone it, downloading zip file is enough. Remember to choose `develop` branch since `master` is really outdated and lacks features.

We also need a broker to operate our MQTT network. You can use [Mosquitto](https://mosquitto.org/) to turn your host into a broker for your local network. Follow their download and installation instructions. Run our broker with `mosquitto` command. If you are on Windows, run the `mosquitto.exe` in PowerShell/Command Prompt instead.

Compile and run the sample codes

Go to `MQTTClient/samples/linux`. Open `hello.cpp` and change hostname to your host's IP address. You can see your host's IP address by running `ifconfig` (on Linux).

![Changing hostname]



Build the program with

```shell
g++ hello.cpp -I ../../src/ -I ../../src/linux -I ../../../MQTTPacket/src
../../../MQTTPacket/src/MQTTPacket.c ../../../MQTTPacket/src/MQTTDeserializePublish.c
../../../MQTTPacket/src/MQTTConnectClient.c ../../../MQTTPacket/src/MQTTSubscribeClient.c
../../../MQTTPacket/src/MQTTSerializePublish.c
../../../MQTTPacket/src/MQTTUnsubscribeClient.c -o hello
```

Run `hello` binary file. You should see this output

![Hello output]

```
Version is 0.300000
Connecting to localhost:1883
MQTT connecting
MQTT connected
rc from MQTT subscribe is 1
Message 1 arrived: qos 0, retained 0, dup 0, packetid 0
Payload Hello World!  QoS 0 message from app version 0.300000
Now QoS 1
Message 2 arrived: qos 1, retained 0, dup 0, packetid 1
Payload Hello World!  QoS 1 message from app version 0.300000
Message 3 arrived: qos 2, retained 0, dup 0, packetid 2
Payload Hello World!  QoS 2 message from app version 0.300000
Finishing with 3 messages received
```

Build the program again but change `g++` to `arm-linux-gnueabihf-g++`. Copy the newly compiled `hello` binary file to your BeagleBone and run it there. Make sure your BeagleBone and your host are on the same network. You should see the same output as above.

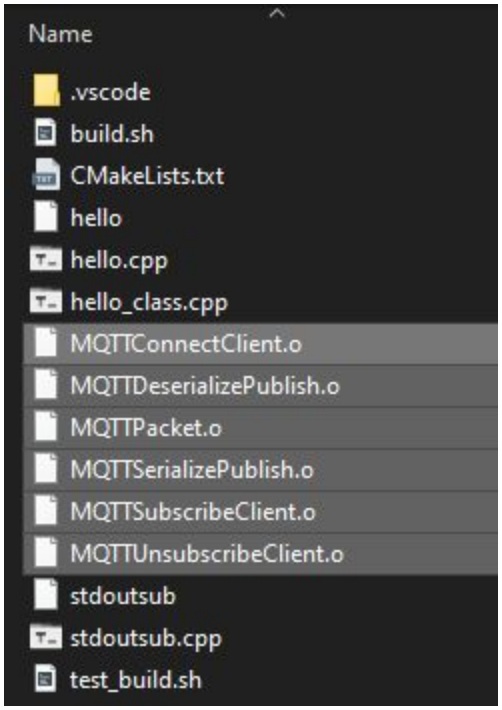
If you get the same output, congratulations! You have properly set up MQTT broker and are ready to use MQTTClient in your project.

Use MQTTClient in your project

Still in the sample linux folder, run this shell command to gather all the necessary objects. These objects are for compilation for your BeagleBone.

```shell
arm-linux-gnueabihf-g++ -c ../../../MQTTPacket/src/MQTTPacket.c
../../../MQTTPacket/src/MQTTDeserializePublish.c ../../../MQTTPacket/src/MQTTConnectClient.c
../../../MQTTPacket/src/MQTTSubscribeClient.c ../../../MQTTPacket/src/MQTTSerializePublish.c
../../../MQTTPacket/src/MQTTUnsubscribeClient.c
```

![Compiled objects]



Copy these objects to a known location. Let's call path to this location `<objects>`.

Also remember the paths for `MQTTClient/src` and `MQTTPacket/src`. You will need them to compile your project.

Include these lines in your header file.

```cpp
#include <memory.h>
#include "MQTTClient.h"
#include "linux/linux.cpp"
#define MQTTCLIENT_QOS2 1
#define DEFAULT_STACK_SIZE -1
```

Now you can use MQTTClient in your project. Consult `hello.cpp` and `hello_class.cpp` in samples folder for how to use the library.

In order to compile your project, you need to declare custom include flags and all the objects you compile before. A typical compilation process looks like this:

```shell
# Remember to replace all the <...> with actual values
MQTT_OBJS=$(shell find <objects> -name '*.o')
```

arm-linux-gnueabihf-g++ <your_cpp_files> $MQTT_OBJS -I <MQTTClient/src> -I <MQTTPacket/src> -o <out_binary_name>
```

**## Some notes on MQTTClient**

* As mentioned before, you need to know broker's IP address in order to join the network.
* Client's ID has to be unique for each device. Generate a random string for each device. This string is only for broker to distinguish between clients. Clients will not be able to see others' IDs.
* You have to call method `yield()` in order to receive messages. The library only listens to messages during the duration specified when calling this method.
* In order to maintain the connection, `yield()` has to be called in interval shorter than broker's `KeepAlive` duration. For example, if broker's keep-alive is 10 seconds then `yield()` has to be called every x<10 seconds. For Mosquitto, the default keep-alive duration is 10 seconds(?). Calling `yield()` every 2 seconds works for us. We recommend repeatedly calling `yield()` in a separate thread to maintain the connection.

```cpp
auto yielding = std::thread([&](){
    while (client.isConnected()){
        client.yield(2000);
    }
});
```
* Refer to [MQTT specification](http://mqtt.org/documentation) for further information regarding acceptable topic's names, QoS levels and more.
```