# Building and Running from start.

There are a few required things needed to build the code given.

1. GCC 7.0 or greater, or a compiler that supports C++17
2. QT Library 5.10 or greater
3. QTMQTT from the same or older version of the QT Library you built.
4. Mosquito MQTT
5. Download the code from <https://github.com/Undertoe/CPE647Project>
6. Three pieces of code are needed: the
   1. MQTT Server
   2. The Sensor code.
   3. iPhone Application

The following directions are primarily for a linux standpoint and, in particular, for the raspberry Pi.

# Raspberry Pi Initialization / requirements

First off, this code has been proven on the Raspberry Pi3 B+. This one has a faster processor. This may run on older versions / less powerful versions, but this is not 100% confirmed.

Recommended items for the Pi:  
1) 2.4amp USB inverter

2) some sort of enclosure with a fan for the pi.

3) Recommended SD card: <https://www.amazon.com/gp/product/B06XWN9Q99/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&psc=1>

The SD card is highly important, running with an XC instead of HC is that your pi will run super slughish. Make sure you get a nice high-speed card.

# GCC 7.0

This was the longest step to setup. A script to download and install GCC 8.3 can be found here: <https://www.raspberrypi.org/forums/viewtopic.php?t=212636>

This script will grab and install everything. Process will likely take 8+ hours so its recommended that you start this at night and come back to it in the morning. After the make command, the make install will run for another hour or so before being done. After this point, C++17 compatible code can be built from command line similar to the following:

g++ file.cpp -o output -std=c++1z

# QT Library

QT Library must be built from source. See this thread for initial information: <https://www.tal.org/tutorials/building-qt-512-raspberry-pi>

Most steps were followed, but the following modification must be made:

* Do not run with the opencl modules.
* Change the second option line (after -v) to “-no-opencl \”

This will appear to “fail” but it builds the library enough for our use. Our software does not run with a GUI so we don’t need to worry about too much. GUI was tested with this build and it seems to work fine.

After this, install qtcreator with the following:

“sudo apt-get install qtcreator”

Launch qtcreator from the terminal or in the menu -> Programming -> qtcreator

From here you now need to set up the build chain.

1. In QT Creator, select the Tools dropdown, and open Options.
2. Select the Build and Run
3. On Compilers:
   1. Click Add-> GCC -> C++
   2. Click Browse
   3. Navigate to the following: “/usr/local/bin” and find g++
   4. Do the same with C and use gcc instead of g++
   5. Give these a name that is unique, recommended: gcc8.3 and g++8.3
4. On Qt Versions:
   1. Click Add
   2. Navigate to /opt/Qt5.12/bin/ and select the qmake
5. On Kits:
   1. Click Add
   2. Give this a name (Recommended: “QT5.12.3”)
   3. Make sure “Desktop” is selected on Device Type
   4. Make sure “Local PC” is selected for Device.
   5. Select your newly made compilers for C: and C++: compilers
   6. Select your Qt Version you made above
6. Click apply

Your QT Build should now be set up.

# QTMQTT

This will install the QT MQTT package.

Some background: This package is part of the QT Embedded devices build, and when your projects are being sold, you MUST buy a license to build and use these packages. If your project is opensource, you are welcome to build and use this software with proper licenses, but you will have to manually build it yourself.

First start with downloading the software. From a terminal type:  
*“git clone git://code.qt.io/qt/qtmqtt.git”*

Now you must open QT creator and allow qmake to set these up manually. There is a way to do this from the terminal but through QT Creator is actually quicker and easier.

When you open the project, set up two ghost build directories for these, recommended is

“../QTMQTT\_Build/Debug/” and “../QTMQTT\_Build/Release/” These directories are relative from the .pro that you are building.

To set up a ghost build, after the project is opened, open the “Project” tab on the left (it has an image that appears to be a wrench) and click on the checkbox that says “Shadow build:” Put those directories in the build directories. You will likely have to provide an absolute path.

Build both Debug and Release. Then from a terminal navigate to the Debug directory. Type the following:  
“make clean”

“make” or “make -j3” (this will build it faster)

“make install”

Do the same for the Release directory.

This will now add the module to QT’s QMake so that it can be linked to. You can also now build the sample programs individually from QT Creator.

# Mosquito MQTT

Download and install mosquito MQTT by using:  
*“sudo apt-get install mosquito”*

Run it with:

*“mosquito”*

It is normally set to start on boot.

# Server Code Install

Download the entire code from Github:

<https://github.com/Undertoe/CPE647Project>

This will come with the Sensor code, server code, and the iPhone code.

For the server: open QT Creator and select the .pro for the QT Creator project.

You will need to complete a few string variables. Most of these are located in

“gclouddata.h” These strings are URL links that you must replace <PROJECT\_ID> with firebase project url, and you have to add your own string files to the server.

After these are set,the project should build and run with minimal to 0 issues.

## Adding a new sensor

Adding a new sensor to the project should be relatively easy.

1. Implement a new sensor. This must be done similar to the Sensor/imusensor.h file. You will need to implement your own data file and implement the various override functions done in the imu sensor.
2. Implement the various new enumeration and string conversions for the new enum for the new sensor type. this enum should be found in “enum class SensorType” from “sensorinfo.h”.
3. Currently, implementation for printing the data is handled in the Processor:: EndOfThreadDSListenerRun(). Its recommended future implementors to bring that implementation to the individual sensors instead of inside of this run code.
4. DataStore::Subscribe() needs to have the new sensor added to its implementation.

After these 4 checkboxes are done, this this implementation of the server SHOULD be able to handle a new sensor quickly.

# iPhone Code

**NOTE: YOU MUST HAVE A MAC TO BUILD AND RUN THIS CODE. IT MUST BE RUNNING THE MOST RECENT VERSION OF XCODE. YOU WILL ALSO NEED AN IPHONE TO BUILD AND RUN THIS. YOU WILL ALSO NEED AN ACTIVE APPLE DEVELOPER ACCOUNT FOR PUSH NOTIFICATION SERVICES TO WORK WITH YOUR SOFTWARE (100$). This takes about 24 hours to process and be approved.**

This part is a bit complicated. Follow the directions here: <https://www.appcoda.com/push-notification-ios/> as these work.

From here you must follow the google cloud Firebase setup:

<https://firebase.google.com/docs/cloud-messaging/ios/client>

<https://firebase.google.com/docs/cloud-messaging/ios/certs>

<https://firebase.google.com/docs/cloud-messaging/ios/first-message>

From here you can run a sample script to send a notification. You will need to replace the following items:  
<FIREBASE\_KEY>: this is found from the following method in the iOS code:

A screenshot of a cell phone

Description automatically generated

<DEVICE\_KEY> can be found in this piece of code: 

Device Keys can be arrayed together as needed if multiple devices are going to messages. The script is as follows:

curl -X POST

--header "Authorization: key=<FIREBASE\_KEY>"

--header "Content-Type: application/json"

https://fcm.googleapis.com/fcm/send

-d "{\"to\":\"<DEVICE\_KEY>=\",\"notification\":{\"body\": \"HEY YO\"}}"