Hugh Johnson

hljohnso@purdue.edu

CNIT 546

Lab 1

September 18, 2024

In section 3 of the lab, we were given source code to an ESP8266 project that reports temperature and humidity to a ThingsBoard dashboard. We were given source code for the MCU and a dashboard to import into our ThingsBoard account. All that was supposed to be required of us was to connect three pins on the ESP8266 to the DHT22, change the SSID and password for the WIFI, and put our individualized device token in the source code.

The source code that we were given was several years old. The syntax for the ThingsBoard library and other required libraries was deprecated. The source code would not compile with the newest Arduino IDE, using the newest ESP8266 board software or the libraries.

I could either work backwards and try and figure out what old versions of software I needed to get the source code to compile, or I could simply ensure that I had the newest software installed and rewrite the project. I chose the latter.

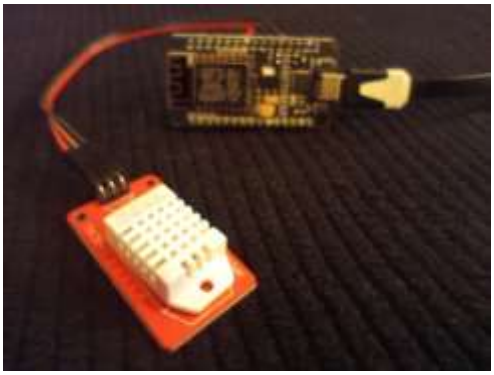
Unlike the author of the original code, I included versions of each of the libraries in comments at the top of my source code for posterity. This enables folks duplicating my setup to successfully compile my code.

When trying to figure out how to communicate with ThingsBoard, I could not find what I consider sufficient API documentation for ThingsBoard, but ThingsBoard does provide up-to-date examples on Github. I used the source code at the link below as an example of how to communicate w/ ThingsBoard from the ESP8266 via MQT.

https://raw.githubusercontent.com/thingsboard/thingsboard-client-sdk/master/examples/0003-esp8266_esp32_send_data/0003-esp8266_esp32_send_data.ino

The physical setup of the project was simple. The DHT22 sensor connected to the MCU via three pins: 3.3v, ground, and a data pin. The ESP8266 connected to the PC via USB. USB supplies 5v power to

the MCU which gets stepped down to 3.3v as per the chip's power requirements. USB is also used as a two-way data connection from PC to MCU.



Data displayed as required on the ThingsBoard dashboard.

The section 4 of the lab was similar to section 3, with a little more complexity added. This part of the lab used an ESP32 and DHT22 sensor, but also had two groups of three LEDs. One group of LEDs were to be controlled (on-off) by virtual dip switches on a ThingsBoard dashboard, while the other group flashed at a speed controlled by a virtual knob on that same dashboard.

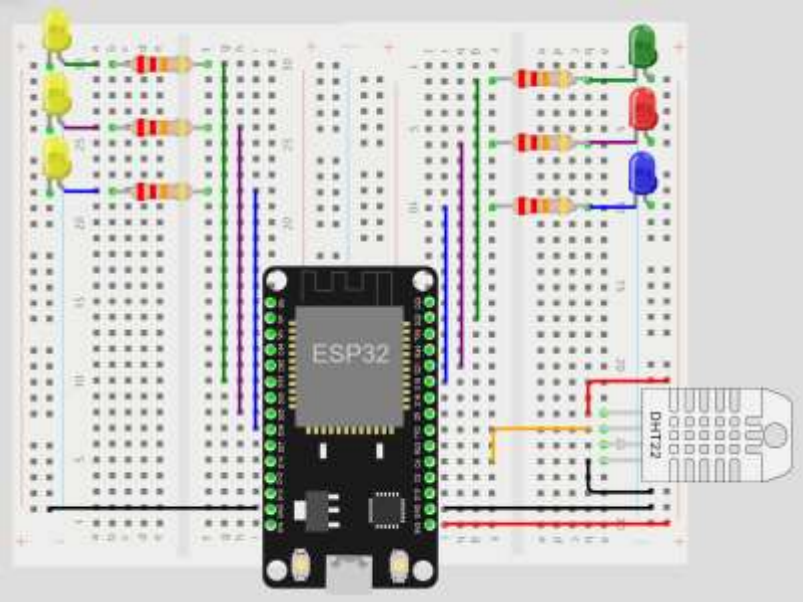
I had the same issue with software and library versions on this portion of the lab. I tackled the problem the same way. I used the example at this Github link for inspiration:

https://raw.githubusercontent.com/thingsboard/thingsboard-client-sdk/master/examples/0010-esp8266_esp32_rpc/0010-esp8266_esp32_rpc.ino

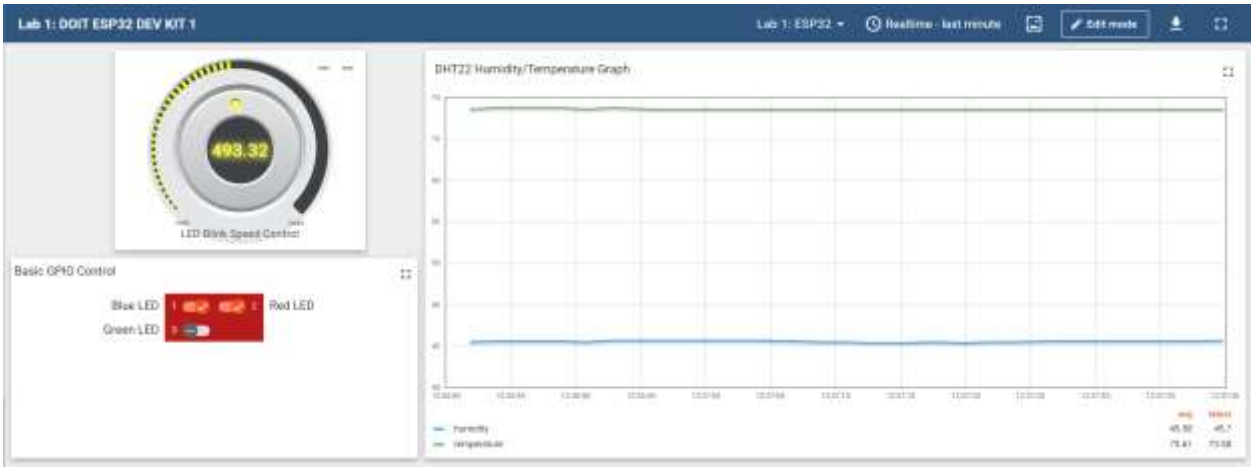
All one is really doing is using GETTERS and SETTERS via MQT remote procedure calls, ThingsBoard to ESP32, to control the lights.

Again, I documented the versions of software and libraries at the top of the source code for the project.

I used WOKWI diagram the project: https://wokwi.com/projects/407671274027447297



ThingsBoard Dashboard:



Since I changed the name of the GETTERS and SETTERS in my code, I had to make the respective change on the ThingsBoard API as shown on the images below:

Basic GPIO Control

Basic GPIO Control

Data | Appearance | Widget card | Actions | Mobile

Preview | Decline | Apply

Data settings

"No data to display" alternative message: Set message

Panel settings

Background color*: #b71c1c

GPIO switches

- Blue LED (pin:1) - [row:0) - [col:0]
- Red LED (pin:2) - [row:0) - [col:1]
- Green LED (pin:3) - [row:1) - [col:0]

Add GPIO switch

RPC settings

RPC request timeout (ms)*: 10000

GPIO status request

Method name*: getLedStatus

```
{ }
```

GPIO status change request

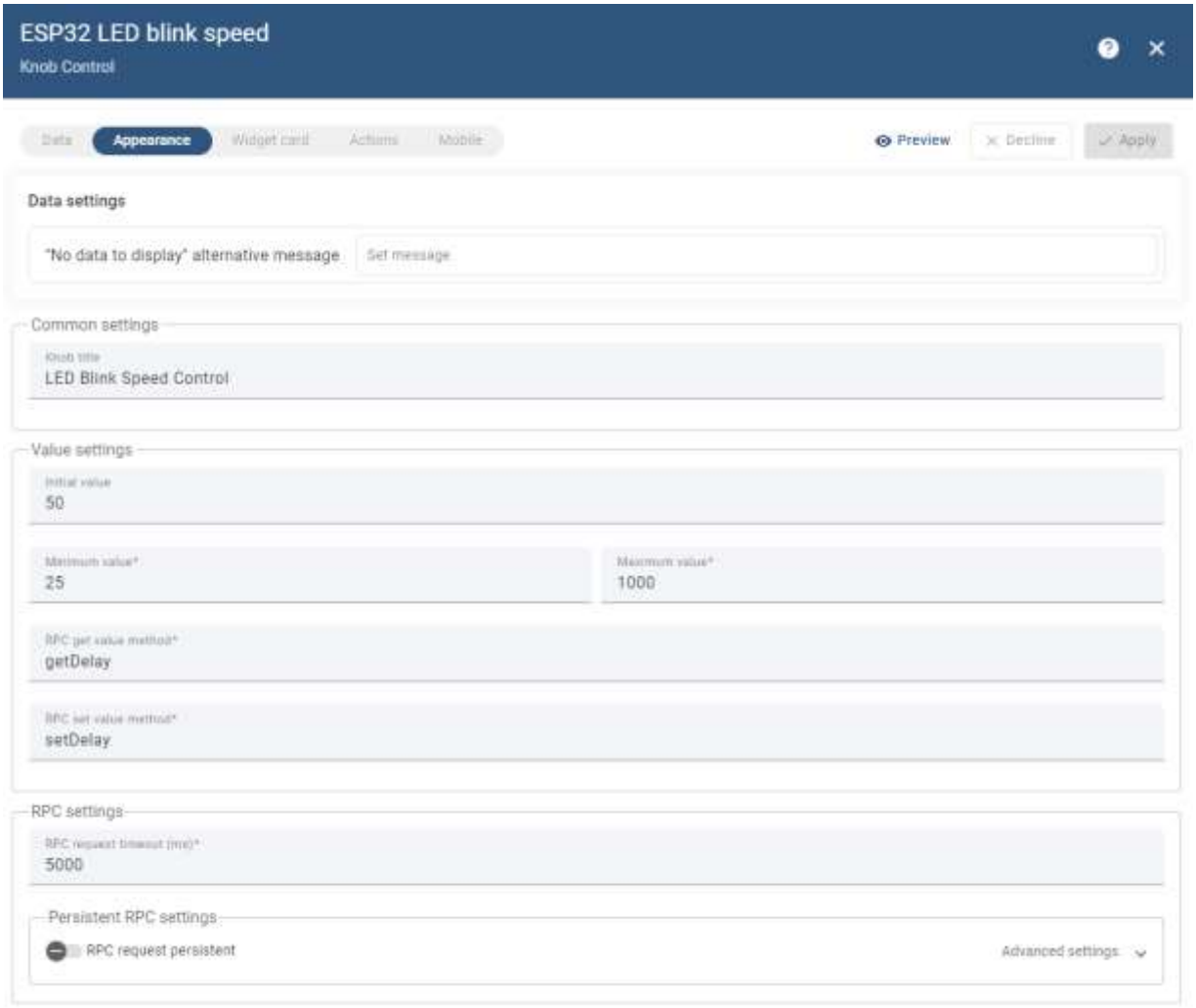
Method name*: setLedStatus

```
{
   "pin": "{$pin}",
   "enabled": "{$enabled}"
}
```

Parse gpio status function: f(body, pin) *

Tidy

```
//return body[pin] === true;
return body["json_data"][pin] === true;
```


ESP32 LED blink speed
Knob Control
Data
Appearance
Widget card
Actions
Mobile
Preview
Decline
Apply
Data settings
"No data to display" alternative message
Set message
Common settings
Knob title
LED Blink Speed Control
Value settings
Initial value
50
Minimum value*
25
Maximum value*
1000
RPC get value method*
getDelay
RPC set value method*
setDelay
RPC settings
RPC request timeout (ms)*
5000
Persistent RPC settings
RPC request persistent
Advanced settings

Conclusion:

ThingsBoard does not document their API well, though they do provide up-to-date examples on github which are more than sufficient for an experienced programmer. Those with less experience will struggle as demonstrated by our class. ThingsBoard is a Ukrainian company which is involved in a major war and has political turmoil. Their demo site, which we use, is hosted in the Google Cloud, so as far as I know has not been affected. As an IT professional with decision making responsibilities, I would not choose TB to provide any services for my organization. That could be career suicide. There are too many other choices available. With that said, I do use another Ukrainian product, PlatformIO, an Open Source plugin for Visual Studio Code to develop software for IoT. Though I don't use PlatformIO in a professional setting, I believe the use case is different enough where it could be used on commercial products. I would probably choose a home grown dashboard system, unless my company were already aligned with large technology services provider.

Source code: https://github.com/khmerhugh/Lab1-IoT