Chapter 8: Class Project

Objective

Time: 2 ¾ Hours

Fundamentals

Exercise(s)

Your project is to build an IoT weather station. Your weather station *thing* will have four state variables:

1. "temperature" (float)
2. "humidity" (float)
3. "weatherAlert" (true or false)
4. "IPAddress" (ipv4 4dot syntax)

The behavior of the weather station will be:

1. Measure local temperature and humidity. This information can be read from the analog co-processor shield kit using I2C (see the I2C exercises in the peripherals chapter).
    a. Read the values on a regular basis (e.g. every 100ms).
2. Monitor the four CapSense buttons. Their eventual functions will be:

CapSense Button	Function
0	Return to the display of the local information (see step 3)
1	Scroll through the data for each available *thing* (see step 8)
2	Toggle the weather alert for your *thing* and publish it (see steps 3 and 7)
3	Publish the temperature and humidity data for your *thing* (see step 7)

    Hint: You can read the CapSense button values at the same time that you read the temperature and humidity values.
    Hint: You will most likely need a mutex to make sure only one thread is accessing the I2C bus at a time.
    Hint: Don't forget to use __attribute__((packed)) if you have an I2C buffer that isn't all 32-bit values. See the I2C section of the peripherals chapter for details.
3. Display the state of your device on the OLED screen. Only refresh the screen for temperature and humidity only if one of the values has changed. You should use 4 lines for the display:
    a. Your thing name (this will be assigned to you – see step 4)
    b. Your device's IP address
    c. Temperature
    d. Humidity
    e. Put a "*" next to the *thing* name on the first line when the weatherAlert is set to true. Refresh the display as soon as its value changes.
4. Connect to a provided MQTT broker:

amk6m51qrxr2u.iot.us-east-1.amazonaws.com

5. Your *thing* name will be "ww101_<nn>" where <nn> will be a number assigned to you. For example ww101_01.
6. The credential and private key for your *thing* can be found in the class material folder.
    a. Hint: After updating the key files, you should run a "Clean" on the project. Otherwise, the project will not see the new keys.
7. Update the state of your *thing* on the broker.
    a. You should:
        i. Publish the IPAddress at startup.
        ii. Publish the weatherAlert when CapSense button 2 is pressed.
        iii. Publish the temperature and humidity when CapSense button 3 is pressed.
        iv. Publish the temperature and humidity automatically every 30 seconds.

    b. Hint: The starting (empty) shadow for your *thing* will look like the following. You will publish JSON messages to the *thing* shadow to provide updates.

Shadow state:

```
{
  "reported": {
    "temperature": 0,
    "humidity": 0,
    "weatherAlert": false,
    "IPAddress": "0.0.0.0"
  }
}
```

You can use the sprintf function to create the JSON messages. Remember that spaces and carriage returns are not required. Also remember that quotation marks in the message must be escaped with a \ character. For example, to create a JSON message to send the temperature from the structure psoc_data.temperature, you could do something like this:

```
char json[128];

sprintf(json, "{\"state\" : {\"reported\" : {\"temperature\":%.1f} } }", psoc_data.temperature);
```

Make sure the array you use to hold the message is large enough. If it isn't you could get very unpredictable results.

8. Subscribe to receive updates from the other *things* that have been assigned for the class.
    a. When CapSense button 1 is pressed, scroll through the assigned things and display their data using the same format as your local thing's display (see step 2).
    b. Hint: It is easiest to just maintain a list of all of the *things* that have been assigned for the class (i.e. ww101_01, ww101_02, etc.)
9. Add a serial terminal interface to implement the following commands (see UART exercises in the peripherals chapter):

    t – Print temperature and publish

h – Print humidity and publish
A – Publish weather alert ON
a – Publish weather alert OFF
S – Turn subscriptions for other things ON
s – Turn subscriptions for other things OFF
P – Turn printing of messages from other *things* ON
p – Turn printing of messages from other *things* OFF
l – Scan for all *things* with valid weather data and print the list
x – Print the current known state of the data from all *things*
c – Clear the terminal and set the cursor to the upper left corner
? – Print the list of commands

It would be cool if you:

1. Used threads
    a. As an example, you might have threads that:
        i. Read I2C data from the PSoC
        ii. Update the OLED display
        iii. Perform the UART command interface functions (both input and output)
        iv. Publish data to the Cloud
        v. Subscribe to specific things/topics from the Cloud
    b. Use RTOS functions to control the interaction between threads. For you could:
        i. Use a MUTEX to make sure the I2C operations to the PSoC and OLED display don't collide
        ii. Use semaphores or queues to control when the display is updated, when publishing is done, what data is published, etc.
        iii. Use a timer to cause temperature and humidity to be published every 30 seconds.
2. Used the linked_list library to maintain a local database of *thing* data
3. Used VT100 escape codes to make a pretty screen:
    a. http://ascii-table.com/ansi-escape-sequences-vt-100.php
4. Used the console library functions to build the UART interface
5. Used the DCT to write the configuration
6. Created an HTTP server to display all of the information