Appendix B: Master Code

Contents:

1. Indoor Blinds

```
/*************************************************************
 Authors: Blind Me With SciEEnce
          Caitlin Gruis, Enrick Hinlo, and Christopher Ravasio
 Date: Spring 2016
 Purpose: Servo Motor Blinds Code
 Description: This code is for a set of automated blinds. It is
connected to an online MQTT server. It moves a set of blinds
either based on light data from a weather sensor, or manual input
from a user via a phone app.
*************************************************************/

/* Include Libraries */
#include <FS.h>
#include <ESP8266WiFi.h>
#include <DNSServer.h>
#include <PubSubClient.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <Servo.h>

Servo myservo; // create servo object to control a servo

char incomingByte; // for incoming serial data

int blindsPrevious; //the last position of the blinds

int lightListen = 0; //determines if blinds are in automatic mode
int autoPosition = 0; //position of blinds in auto mode


/************************ WiFi Access Point
*******************************/

#define WLAN_SSID       "ND-guest"
#define WLAN_PASS       ""

/************************ Adafruit.io Setup
*******************************/

#define SERVER_ADDRESS      "senior-mqtt.esc.nd.edu"  // Amazon
MQTT Web Server

#define SERVER_PORT         1883        // standard port


void callback(const MQTT::Publish& pub) {

  Serial.println(pub.payload_string());
```

```
  Serial.println();

  /* If blinds receive a "yes" from automation, listen to the
light sensor */
  if(pub.topic() == "sciEEnce/automation"){
    if (pub.payload_string() == "yes"){
      lightListen = 1;
      Serial.println(lightListen);
    }else if (pub.payload_string() == "no"){
      lightListen = 0;
    }
  }
  /* If automation is on, this sets the blinds position based on
the light sensor */
  if((pub.topic() == "sciEEnce/light") && (lightListen == 1)){

    String lightString = pub.payload_string();
    int lightLevel = lightString.toInt();

    if (lightLevel <= 30){
      if(autoPosition !=20){
        autoPosition = 20;
        myservo.write(autoPosition);
      }
    }else if((lightLevel >30) && (lightLevel < 300)){
      if(autoPosition !=100){
        autoPosition = 100;
        myservo.write(autoPosition);
      }
    }else if(lightLevel >=300){
      if(autoPosition!=60){
        autoPosition = 60;
        myservo.write(autoPosition);
      }
    }
  }

  /* If automation is off, this sets the blinds manually based on
user input */
  if ((pub.topic() == "sciEEnce/blinds/allblinds") ||
(pub.topic() == "sciEEnce/blinds/blinds1")){

    String blindsString = pub.payload_string();
    int blindsPosition = blindsString.toInt();

    if (blindsPosition != blindsPrevious) {
      myservo.write(blindsPosition);
      blindsPrevious = blindsPosition;
      delay(10);
    }
  }
}// end of callback function
```

```
// Create an ESP8266 WiFiClient class to connect to the MQTT
server.
WiFiClient wf_client; // instantiate wifi client
PubSubClient client(wf_client, SERVER_ADDRESS); // pass to pubsub

/* Beginning of setup function, runs once */
void setup() {

  myservo.attach(13);  // says the servo is on GPIO13
  myservo.write(20);  // initializes the servo position to 20
degrees

  // Setup console
  Serial.begin(9600);
  delay(10);
  Serial.println();
  Serial.println(F("Modified pubsub client basic code using
modified pubsub software"));
  client.set_callback(callback);

  //WiFiManager
  //Local intialization. Once its business is done, there is no
need to keep it around
  WiFiManager wifiManager;

  //exit after config instead of connecting
  wifiManager.setBreakAfterConfig(true);

  //tries to connect to last known settings
  //if it does not connect it starts an access point with the
specified name
  //here  "AutoConnectAP" with password "password"
  //and goes into a blocking loop awaiting configuration
  if (!wifiManager.autoConnect("Blinds", "blinds")) {
    Serial.println("failed to connect, we should reset as see if
it connects");
    delay(3000);
    ESP.reset();
    delay(5000);
  }

  //if you get here you have connected to the WiFi
  Serial.println("connected...yeey :)");

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

}
```

```
void loop() {


  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
      if (client.connect("Blinds")) {

        // Subscribe to the following MQTT topics
        Serial.println("Connected to MQTT server");
        client.subscribe("sciEEnce/blinds/allblinds");
        client.subscribe("sciEEnce/blinds/blinds1");
        client.subscribe("sciEEnce/automation");
        client.subscribe("sciEEnce/light");
        Serial.println("Just subscribed to blinds1, allblinds,
automation, and light");

      }
    }
/* wait for incoming messages */

    if (client.connected())
      client.loop();
  }
}
```

2. Outdoor Weather Station

```
/*************************************************
 Authors: Blind Me With SciEEnce
          Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
 Date: Spring 2016
 Purpose: Outdoor Weather Station Code
 Description: This code is for an outdoor weather station that
includes a temperature,
              humidity, pressure, and light sensor. It connects
to an online MQTT
              server and publishes data from the weather sensors
to it repeatedly.
 *************************************************/

/* Include Libraries */
#include <FS.h>
#include <ESP8266WiFi.h>
#include <DNSServer.h>
#include <PubSubClient.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <Wire.h>
#include "Adafruit_HTU21DF.h"
#include "Adafruit_MPL3115A2.h"
#include "SparkFunTSL2561.h"


Adafruit_HTU21DF htu = Adafruit_HTU21DF(); // temp/humidity
sensor declaration
SFE_TSL2561 light; // light sensor delcaration

//light sensor variables
boolean gain;     // Gain setting, 0 = X1, 1 = X16;
unsigned int ms;  // Integration ("shutter") time in milliseconds

float tempReading; //for temperature sensor readings
float humidityReading; // for huimdity readings

/************************ WiFi Access Point
*******************************/

#define WLAN_SSID       "ND-guest"
#define WLAN_PASS       ""

/************************ Adafruit.io Setup
*******************************/

#define SERVER_ADDRESS      "senior-mqtt.esc.nd.edu"  // server
in 213 SR
```

```
#define SERVER_PORT         1883        // standard port

void callback(const MQTT::Publish& pub) {
  // handle message arrived
  Serial.print("Message arrived [");
  Serial.print(pub.topic());
  Serial.print("] ");

  Serial.println(pub.payload_string());

  Serial.println();
}// end of callback function

// Create an ESP8266 WiFiClient class to connect to the MQTT
server.
WiFiClient wf_client; // instantiate wifi client
PubSubClient client(wf_client, SERVER_ADDRESS); // pass to pubsub

void setup() {

  // Setup console
  Serial.begin(9600);
  Wire.begin(5,4);
  light.begin();
  gain = 0;
  unsigned char time = 2;
  light.setTiming(gain,time,ms);
  light.setPowerUp();

  delay(10);
  client.set_callback(callback);

  if (!htu.begin()) {
    Serial.println("Couldn't find temp/humidity sensor!");
    while (1);
  }

  //WiFiManager
  //Local intialization. Once its business is done, there is no
need to keep it around
  WiFiManager wifiManager;

  //exit after config instead of connecting
  wifiManager.setBreakAfterConfig(true);

  //reset settings - for testing
  //wifiManager.resetSettings();

  //tries to connect to last known settings
  //if it does not connect it starts an access point with the
specified name
```

```
  //here  "AutoConnectAP" with password "password"
  //and goes into a blocking loop awaiting configuration
  if (!wifiManager.autoConnect("WeatherStation", "weather")) {
    Serial.println("failed to connect, we should reset as see if
it connects");
    delay(3000);
    ESP.reset();
    delay(5000);
  }

  //if you get here you have connected to the WiFi
  Serial.println("connected...yeey :)");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {

  if (WiFi.status() == WL_CONNECTED) {
    if (!client.connected()) {
     reconnect();
    }
  }

  if (client.connected())
    client.loop();
    delay(ms);

    // Get weather readings
    tempReading = (htu.readTemperature()*1.8) + 32;
    humidityReading = htu.readHumidity();

    unsigned int data0, data1;
    double lux;    // Resulting lux value

    if (light.getData(data0,data1)){
    boolean good;  // True if neither sensor is saturated

    // Perform lux calculation:
    good = light.getLux(gain,ms,data0,data1,lux);

    // Print out the results:
    Serial.print("Lux: ");
    Serial.println(lux);
  }
  else
  {
    // getData() returned false because of an I2C error, inform
the user.
    byte error = light.getError();
    printError(error);
```

```
  }

    Serial.print("Temperature: ");
    Serial.print(tempReading);
    Serial.println(" F");
    Serial.print("Humidity: ");
    Serial.print(humidityReading);
    Serial.println("%");
    Serial.println();

    // Convert weather readings to strings so MQTT server can
publish data
    String lightString = String(double(lux));
    String temperatureString = String(int(tempReading));
    String humidityString = String(int(humidityReading));

    // Publish to MQTT server

client.publish(MQTT::Publish("sciEEnce/weatherStation1/light",
lightString).set_qos(1));
    client.publish(MQTT::Publish("sciEEnce/weatherStation1/temp",
temperatureString).set_qos(1));

client.publish(MQTT::Publish("sciEEnce/weatherStation1/humidity",
humidityString).set_qos(1));

    //Put ESP to sleep for a given amount of time
    ESP.deepSleep(10 * 1000000); //ESP will deep sleep for 5
seconds
}

void reconnect() {
  // Loop until we're reconnected
  delay(5);
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("Weather Station")) {
      Serial.println("connected");
    } else {
      Serial.println(" try again in 1.5 seconds");
      // Wait 1.5 seconds before retrying
      delay(1500);
    }
  }
  delay(500);

}

void printError(byte error)
  // If there's an I2C error, this function will
  // print out an explanation.
```

```
{
  Serial.print("I2C error: ");
  Serial.print(error,DEC);
  Serial.print(", ");

  switch(error)
  {
    case 0:
      Serial.println("success");
      break;
    case 1:
      Serial.println("data too long for transmit buffer");
      break;
    case 2:
      Serial.println("received NACK on address (disconnected?)");
      break;
    case 3:
      Serial.println("received NACK on data");
      break;
    case 4:
      Serial.println("other error");
      break;
    default:
      Serial.println("unknown error");
  }
}
```

3. Mobile Application

ViewController.swift:

```
//
//  ViewController.swift
//  Authors: Blind Me With SciEEnce
//           Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
//

import UIKit
import CocoaMQTT

var pickerData : NSMutableArray = ["All Blinds","Bedroom Blinds",
"Kitchen Blinds"]
var topicData : NSMutableArray = ["allblinds", "blinds1", "blinds2"]

class ViewController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {
    var mqtt: CocoaMQTT?
    var whichBlinds = String()
    var i = Int()


    @IBOutlet weak var blindsPicker: UIPickerView!
    @IBOutlet weak var blindsSlider: UISlider!
    @IBOutlet weak var automationButton: UISegmentedControl!


    @IBAction func blindsMover(sender: AnyObject) {
            mqtt?.publish("sciEEnce/blinds/" +  whichBlinds,
withString: String(Int(self.blindsSlider.value)))
    }

    @IBAction func automationChanged(sender: AnyObject) {
        switch automationButton.selectedSegmentIndex
        {
        case 0:
            mqtt?.publish("sciEEnce/automation", withString: "no")
            blindsSlider.enabled = true
            blindsPicker.userInteractionEnabled = true
        case 1:
            mqtt?.publish("sciEEnce/automation", withString: "yes")
            blindsSlider.enabled = false
            blindsPicker.userInteractionEnabled = false
        default:
            break;
        }
    }



    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        whichBlinds = "allblinds"

        self.blindsPicker.delegate = self
        self.blindsPicker.dataSource = self

        mqttSetting()
        mqtt!.connect()

    }

    func mqttSetting() {
        let clientIdPid = "SDTest" +
String(NSProcessInfo().processIdentifier)
        mqtt = CocoaMQTT(clientId: clientIdPid, host: "senior-
mqtt.esc.nd.edu", port: 1883)
        //mqtts
        //let mqtt = CocoaMQTT(clientId: clientIdPid, host:
"localhost", port: 8883)
        //mqtt.secureMQTT = true
        if let mqtt = mqtt {
            mqtt.username = "ND-guest"
            mqtt.password = ""
            mqtt.willMessage = CocoaMQTTWill(topic: "/will", message:
"dieout")
            mqtt.keepAlive = 90
            mqtt.delegate = self
        }
    }


    // The number of columns of data
    func numberOfComponentsInPickerView(pickerView: UIPickerView) ->
Int {
        return 1
    }

    // The number of rows of data
    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent
component: Int) -> Int {
        return pickerData.count
    }

    // The data to return for the row and component (column) that's
being passed in
    func pickerView(pickerView: UIPickerView, titleForRow row: Int,
forComponent component: Int) -> String? {
        return pickerData[row] as? String
    }

    func pickerView(pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int)
    {
        whichBlinds = topicData[row] as! String
    }
```

```
}


    extension ViewController: CocoaMQTTDelegate {

        func mqtt(mqtt: CocoaMQTT, didConnect host: String, port: Int)
{
            print("didConnect \(host):\(port)")
        }

    func mqtt(mqtt: CocoaMQTT, didConnectAck ack: CocoaMQTTConnAck) {
        //print("didConnectAck \(ack.rawValue)")
        if ack == .ACCEPT {
            mqtt.subscribe("sciEEnce/" + whichStation + "/temp", qos:
CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/humidity",
qos: CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/pressure",
qos: CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/light", qos:
CocoaMQTTQOS.QOS1)
            mqtt.ping()

        }

    }

    func mqtt(mqtt: CocoaMQTT, didPublishMessage message:
CocoaMQTTMessage, id: UInt16) {
        print("didPublishMessage to \(message.topic) with message:
\(message.string)")
    }

    func mqtt(mqtt: CocoaMQTT, didPublishAck id: UInt16) {
        //print("didPublishAck with id: \(id)")
    }

        func mqtt(mqtt: CocoaMQTT, didReceiveMessage message:
CocoaMQTTMessage, id: UInt16 ) {
            print("didReceivedMessage: \(message.string) from topic
\(message.topic)")

            if (message.topic == ("sciEEnce/" + whichStation +
"/temp")){
                lastTemp = message.string
            }else if (message.topic == ("sciEEnce/" + whichStation +
"/humidity")){
                lastHumidity = message.string
            }//else if (message.topic == "sciEEnce/" + whichStation +
"/pressure"){
                // pressureLabel.text = message.string
                // lastPressure = message.string
            /*}*/else if (message.topic == "sciEEnce/" + whichStation +
"/light"){
```

```
                var lightValue: Int?
                lightValue = Int(message.string!)
                if (lightValue <= 3){
                    lastWeather = "Night"
                    lastPic = nightImage
                }else if ((lightValue > 3) && (lightValue <= 800)){
                    lastWeather = "Cloudy"
                    lastPic = cloudyImage
                }else if ((lightValue > 800) && (lightValue <= 1500)){
                    lastWeather = "Partly Cloudy"
                    lastPic = partCloudyImage
                }else if (lightValue > 1500){
                    lastWeather = "Sunny"
                    lastPic = sunnyImage
                }
            }
        }

    func mqtt(mqtt: CocoaMQTT, didSubscribeTopic topic: String) {
        print("didSubscribeTopic to \(topic)")
    }

    func mqtt(mqtt: CocoaMQTT, didUnsubscribeTopic topic: String) {
        print("didUnsubscribeTopic to \(topic)")
    }

    func mqttDidPing(mqtt: CocoaMQTT) {
        print("didPing")
    }

    func mqttDidReceivePong(mqtt: CocoaMQTT) {
        _console("didReceivePong")
    }

    func mqttDidDisconnect(mqtt: CocoaMQTT, withError err: NSError?) {
        _console("mqttDidDisconnect")
    }

    func _console(info: String) {
        print("Delegate: \(info)")
    }

}
```

MQTT Controller.Swift:

```
//
//  MQTTController.swift
//  Authors: Blind Me With SciEEnce
//      Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
//  Date: Spring 2016
//

import UIKit
import CocoaMQTT

class MQTTController: UIViewController {
    var mqtt: CocoaMQTT?

    override func viewDidLoad() {
        super.viewDidLoad()
        mqttSetting()
        mqtt!.connect()

    }

    func mqttSetting() {
        let clientIdPid = "SDTest" +
String(NSProcessInfo().processIdentifier)
        mqtt = CocoaMQTT(clientId: clientIdPid, host: "senior-
mqtt.esc.nd.edu", port: 1883)

        if let mqtt = mqtt {
            mqtt.username = "ND-guest"
            mqtt.password = ""
            mqtt.willMessage = CocoaMQTTWill(topic: "/will", message:
"dieout")
            mqtt.keepAlive = 90
            mqtt.delegate = self
        }
    }

}


extension MQTTController: CocoaMQTTDelegate {

    func mqtt(mqtt: CocoaMQTT, didConnect host: String, port: Int) {
        print("didConnect \(host):\(port)")
    }

    func mqtt(mqtt: CocoaMQTT, didConnectAck ack: CocoaMQTTConnAck) {
        if ack == .ACCEPT {
            mqtt.subscribe("sciEEnce/" + whichStation + "/temp", qos:
CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/humidity",
qos: CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/pressure",
qos: CocoaMQTTQOS.QOS1)
```

```
            mqtt.subscribe("sciEEnce/" + whichStation + "/light", qos:
CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/blinds/+", qos: CocoaMQTTQOS.QOS1)
            mqtt.ping()

        }

    }

    func mqtt(mqtt: CocoaMQTT, didPublishMessage message:
CocoaMQTTMessage, id: UInt16) {
        print("didPublishMessage with message: \(message.string)")
    }

    func mqtt(mqtt: CocoaMQTT, didPublishAck id: UInt16) {
        print("didPublishAck with id: \(id)")
    }

    func mqtt(mqtt: CocoaMQTT, didReceiveMessage message:
CocoaMQTTMessage, id: UInt16 ) {
        print("didReceivedMessage: \(message.string) from topic
\(message.topic)")

        if (message.topic == ("sciEEnce/" + whichStation + "/temp")){
            lastTemp = message.string
        }else if (message.topic == ("sciEEnce/" + whichStation +
"/humidity")){
            lastHumidity = message.string
        }//else if (message.topic == "sciEEnce/" + whichStation +
"/pressure"){
            // pressureLabel.text = message.string
            // lastPressure = message.string
        /*}*/else if (message.topic == "sciEEnce/" + whichStation +
"/light"){
            var lightValue: Int?
            lightValue = Int(message.string!)
            if (lightValue <= 3){
                lastWeather = "Night"
                lastPic = nightImage
            }else if ((lightValue > 3) && (lightValue <= 800)){
                lastWeather = "Cloudy"
                lastPic = cloudyImage
            }else if ((lightValue > 800) && (lightValue <= 1500)){
                lastWeather = "Partly Cloudy"
                lastPic = partCloudyImage
            }else if (lightValue > 1500){
                lastWeather = "Sunny"
                lastPic = sunnyImage
            }
        }
    }

    func mqtt(mqtt: CocoaMQTT, didSubscribeTopic topic: String) {
        print("didSubscribeTopic to \(topic)")
    }
```

```
    func mqtt(mqtt: CocoaMQTT, didUnsubscribeTopic topic: String) {
        print("didUnsubscribeTopic to \(topic)")
    }

    func mqttDidPing(mqtt: CocoaMQTT) {
        print("didPing")
    }

    func mqttDidReceivePong(mqtt: CocoaMQTT) {
        _console("didReceivePong")
    }

    func mqttDidDisconnect(mqtt: CocoaMQTT, withError err: NSError?) {
        _console("mqttDidDisconnect")
    }

    func _console(info: String) {
        print("Delegate: \(info)")
    }

}

extension MQTTController: UITabBarControllerDelegate {
    // Prevent automatic popToRootViewController on double-tap of
UITabBarController
    func tabBarController(tabBarController: UITabBarController,
shouldSelectViewController viewController: UIViewController) -> Bool {
        return viewController !=
tabBarController.selectedViewController
    }
}
```

WeatherController.Swift

```
//
//  WeatherController.swift
//  Authors: Blind Me With SciEEnce
//           Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
//  Date: Spring 2016
//

import UIKit
import CocoaMQTT

var lastTemp: String?
var lastHumidity: String?
var lastPressure: String?
var lastWeather: String?
var lastPic: UIImage?

let sunnyImage = UIImage(imageLiteral: "sunny.png")
let cloudyImage = UIImage(imageLiteral: "cloudy.png")
let nightImage = UIImage(imageLiteral: "night.jpg")
let partCloudyImage = UIImage(imageLiteral: "partlycloudy.png")

var whichStation = "weatherStation1"

class WeatherController: UIViewController {

    var mqtt: CocoaMQTT?

    @IBOutlet weak var tempLabel: UILabel!
    @IBOutlet weak var humidityLabel: UILabel!
    @IBOutlet weak var pressureLabel: UILabel!
    @IBOutlet weak var weatherImage: UIImageView!
    @IBOutlet weak var weatherLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        mqttSetting()
        mqtt!.connect()

        tempLabel.text = lastTemp
        humidityLabel.text = lastHumidity
        pressureLabel.text = "101.5"
        weatherLabel.text = lastWeather

        weatherImage.image = lastPic
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func mqttSetting() {
```

```
        let clientIdPid = "SDTest" +
String(NSProcessInfo().processIdentifier)
        mqtt = CocoaMQTT(clientId: clientIdPid, host: "senior-
mqtt.esc.nd.edu", port: 1883)
        if let mqtt = mqtt {
            mqtt.username = "ND-guest"
            mqtt.password = ""
            mqtt.willMessage = CocoaMQTTWill(topic: "/will", message:
"dieout")
            mqtt.keepAlive = 90
            mqtt.delegate = self
        }
    }

}


extension WeatherController: CocoaMQTTDelegate {

    func mqtt(mqtt: CocoaMQTT, didConnect host: String, port: Int) {
        print("didConnect \(host):\(port)")
    }

    func mqtt(mqtt: CocoaMQTT, didConnectAck ack: CocoaMQTTConnAck) {
        //print("didConnectAck \(ack.rawValue)")
        if ack == .ACCEPT {
            mqtt.subscribe("sciEEnce/" + whichStation + "/temp", qos:
CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/humidity",
qos: CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/pressure",
qos: CocoaMQTTQOS.QOS1)
            mqtt.subscribe("sciEEnce/" + whichStation + "/light", qos:
CocoaMQTTQOS.QOS1)
            mqtt.ping()

        }

    }

    func mqtt(mqtt: CocoaMQTT, didPublishMessage message:
CocoaMQTTMessage, id: UInt16) {
        print("didPublishMessage with message: \(message.string)")
    }

    func mqtt(mqtt: CocoaMQTT, didPublishAck id: UInt16) {
        print("didPublishAck with id: \(id)")
    }

    func mqtt(mqtt: CocoaMQTT, didReceiveMessage message:
CocoaMQTTMessage, id: UInt16 ) {
        print("didReceivedMessage: \(message.string) from topic
\(message.topic)")

        if (message.topic == ("sciEEnce/" + whichStation + "/temp")){
```

```
            tempLabel.text = message.string
            lastTemp = message.string
        }else if (message.topic == ("sciEEnce/" + whichStation +
"/humidity")){
            humidityLabel.text = message.string
            lastHumidity = message.string
        }//else if (message.topic == "sciEEnce/" + whichStation +
"/pressure"){
           // pressureLabel.text = message.string
           // lastPressure = message.string
        /*}*/else if (message.topic == "sciEEnce/" + whichStation +
"/light"){
            var lightValue: Int?
            lightValue = Int(message.string!)
            print(lightValue)
            if (lightValue <= 3){
                weatherLabel.text = "Night"
                weatherImage.image = nightImage
                lastWeather = "Night"
                lastPic = nightImage
            }else if ((lightValue > 3) && (lightValue <= 800)){
                weatherLabel.text = "Cloudy"
                weatherImage.image = cloudyImage
                lastWeather = "Cloudy"
                lastPic = cloudyImage
            }else if ((lightValue > 800) && (lightValue <= 1500)){
                weatherLabel.text = "Partly Cloudy"
                weatherImage.image = partCloudyImage
                lastWeather = "Partly Cloudy"
                lastPic = partCloudyImage
            }else if (lightValue > 1500){
                weatherLabel.text = "Sunny"
                weatherImage.image = sunnyImage
                lastWeather = "Sunny"
                lastPic = sunnyImage
            }
        }
    }

    func mqtt(mqtt: CocoaMQTT, didSubscribeTopic topic: String) {
        print("didSubscribeTopic to \(topic)")
    }

    func mqtt(mqtt: CocoaMQTT, didUnsubscribeTopic topic: String) {
        print("didUnsubscribeTopic to \(topic)")
    }

    func mqttDidPing(mqtt: CocoaMQTT) {
        print("didPing")
    }

    func mqttDidReceivePong(mqtt: CocoaMQTT) {
        _console("didReceivePong")
    }
```

```
    func mqttDidDisconnect(mqtt: CocoaMQTT, withError err: NSError?) {
        _console("mqttDidDisconnect")
    }

    func _console(info: String) {
        print("Delegate: \(info)")
    }

}
```

NewBlindsController.swift

```
//
//  NewBlindsController.swift
//  Authors: Blind Me With SciEEnce
//           Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
//  Date: Spring 2016
//

import UIKit

class NewBlindsController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {
    var whichBlinds = String()
    var arrayIndex = Int()

    @IBOutlet weak var removeLabel: UILabel!
    @IBOutlet weak var topicLabel: UILabel!
    @IBOutlet weak var blindsLabel: UILabel!
    @IBOutlet weak var addremoveButton: UISegmentedControl!
    @IBOutlet weak var topicnameText: UITextField!
    @IBOutlet weak var blindsnameText: UITextField!
    @IBOutlet weak var submitButton: UIButton!
    @IBOutlet weak var blindsPicker: UIPickerView!

    @IBAction func segblindsChanged(sender: AnyObject) {
        switch addremoveButton.selectedSegmentIndex
        {
        case 0:
            blindsPicker.hidden = true
            removeLabel.hidden = true
            topicLabel.hidden = false
            blindsLabel.hidden = false
            topicnameText.hidden = false
            blindsnameText.hidden = false
        case 1:
            blindsPicker.hidden = false
            removeLabel.hidden = false
            topicLabel.hidden = true
            blindsLabel.hidden = true
            topicnameText.hidden = true
            blindsnameText.hidden = true
        default:
            break;
        }
    }

    @IBAction func submitbuttonPress(sender: AnyObject) {

        if (addremoveButton.selectedSegmentIndex == 0){
            pickerData.addObject(blindsnameText.text!)
            topicData.addObject(topicnameText.text!)

            blindsnameText.text = ""
            topicnameText.text = ""
```

```
        }
        else if(addremoveButton.selectedSegmentIndex == 1){

            pickerData.removeObjectAtIndex(arrayIndex)
            topicData.removeObjectAtIndex(arrayIndex)

            blindsnameText.text = ""
            topicnameText.text = ""
        }
    }


    override func viewDidLoad() {
        super.viewDidLoad()
        blindsPicker.hidden = true
        removeLabel.hidden = true

        self.blindsPicker.delegate = self
        self.blindsPicker.dataSource = self
}

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    // The number of columns of data
    func numberOfComponentsInPickerView(pickerView: UIPickerView) ->
Int {
        return 1
    }

    // The number of rows of data
    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent
component: Int) -> Int {
        return pickerData.count
    }

    // The data to return for the row and component (column) that's
being passed in
    func pickerView(pickerView: UIPickerView, titleForRow row: Int,
forComponent component: Int) -> String? {
        return pickerData[row] as? String
    }

    func pickerView(pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int)
    {
        whichBlinds = topicData[row] as! String
        arrayIndex = row

    }
}
```

NewWeatherStationController.Swift

```
//
//  NewWeatherStationController.swift
//  Authors: Blind Me With SciEEnce
//           Caitlin Gruis, Enrick Hinlo, Christopher Ravasio
//  Date: Spring 2016
//

import UIKit

class NewWeatherStationController: UIViewController {

    @IBOutlet weak var newstationText: UITextField!
    @IBOutlet weak var submitButton: UIButton!

    @IBAction func submitbuttonPressed(sender: AnyObject) {

        whichStation = newstationText.text!
        newstationText.text = ""
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```