Kettering University

College of Engineering

Department of Electrical and Computer Engineering

Project Report

Course No. & Title: CE-420-01 Microcomputers II

Instructor's Name: Professor Girma Tewolde

Project Title: Plant Life Monitor

Date Submitted: December 14th, 2017

Team Members:

(1) Jeremy Maxey-Vesperman

(2) Zachary Goldasich

INSTRUCTOR SECTION

Comments: ____

____

____

____

Grade: **Team Member 1:** ____________________

Team Member 2: ____________________

Table of Contents

1. Objectives

This project’s goal is to create a monitoring system that is capable of reporting various metrics of a common houseplant. These metrics include light intensity, temperature, and soil moisture. In order to facilitate the reporting process, the device implements an SMTP-based system to communicate with its owner remotely. Furthermore, in order to extend battery life, many power-saving features starting at the hardware level and working up to the software level were implemented.

2. Theoretical Background

Taking care of houseplants is a popular hobby among many people, but actually handling their watering schedules and the like is often demanding--moreso with some species over others. In order to handle this in a manner friendly to home automation, our project is a “life sensor” that monitors the soil moisture, light, and temperature levels of the plant it’s attached to. This is accomplished through the use of a Light-Dependent Resistor (LDR), also referred to as a photoresistor, a Negative Temperature Coefficient (NTC) thermistor, and soil moisture sensor that measures the resistivity of the soil to determine soil electrical conductivity (EC).

LDRs vary their resistance in accordance with light levels in the room[1]. Usually, LDRs are only utilized as a cheap way to differentiate between light and dark. This is mainly due to the large tolerance range that they are manufactured with; A light intensity of 10 lux could be as low as 50kΩ or as high as 100kΩ in the case of the LDR utilized in this project[2]. However, by taking resistance measurements at varying light levels compared against a lux meter and utilizing statistics, a calibrated light intensity sensor can be created. This correlation between resistance and light intensity can be expressed as:

$$L = R^m * 10^b \tag{2.1}$$

Where L represents light intensity in lux, R is resistance of the LDR, and m and b are the slope and intercept of a best fit line equation obtained from the log-log (Lux vs. R) graph of the calibration data[3].

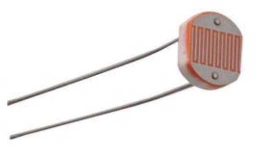


Figure 2.1: Example of a photoresistor

Similarly, NTC thermistors vary their resistance as a response to temperature. This response is negative, meaning that the resistance decreases as temperature increases. There are a few various approaches to correlating temperature to this resistance. One such method utilizes the simplified Steinhart-Hart equation:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{\beta} * ln(\frac{R}{R_0}) \qquad 2.2$$

Where T is current temperature in Kelvin, T_0 is room temperature (K), R is the measured resistance, and R_0 is the resistance the manufactured measured at T_0. This also utilizes a constant, β, that is provided by the sensor manufacturer[4].

Figure 2.2: Example of an NTC thermistor

The driving theory behind soil EC measurement is that there is a strong correlation between how well the soil conducts and its moisture content. Additionally, this measurement can also be used to determine soil particle size as sandier soils have higher resistances[5].

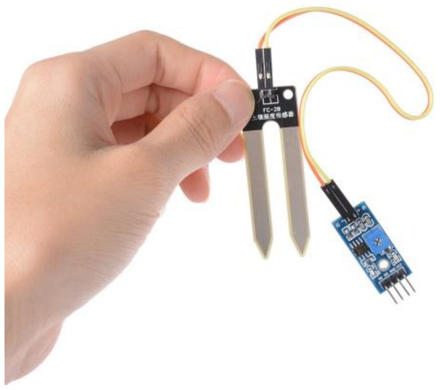


Figure 2.3: Example of a soil moisture sensor

There are several major goals to be discussed here when speaking on the Plant Life Monitor's design:

- It must have a relatively small footprint in order to be unobtrusive
- It must implement as many power saving strategies as is reasonably possible without compromising functionality, as a plant monitor that constantly needed recharging would defeat the purpose
- It must consist of parts that total up to a relatively low price point. The justification for this is twofold:
    - A lower total expense makes the prototyping stage easier to work with since the bill of materials isn't overwhelming.
    - Plant moisture sensing is something of a niche market, but one that remains untapped for one major reason: expense. Most commercial devices out on the market today tend to abuse the fact that the "Internet of Things" is a fad and drive up their prices accordingly. By keeping the pricing of the project's components relatively low, we believe that the

device stands a real chance at producing consumer demand by undercutting the (overpriced) competition.

3. Hardware Design

The following schematic provides an overview of the hardware design for the project:

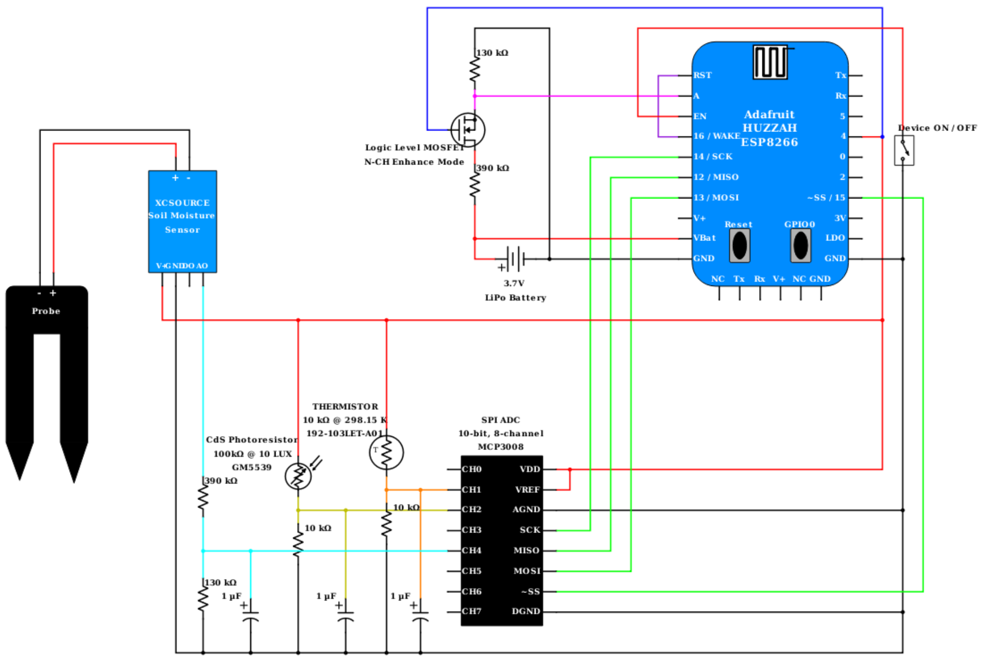


Figure 3.1: Hardware Schematic

The primary component controlling the entire device is an Adafruit HUZZAH ESP8266--a small, inexpensive 80MHz (160 MHz overclockable) Harvard architecture microcontroller unit. The MCU has a total of 80kB Data RAM, 35kB Instruction RAM, and 4MB SPI flash. It also contains an RTC module with a small amount of RAM to hold WiFi connection information and other variables that the programmer would like to store between Deep Sleep cycles, as this remains powered during Deep Sleep. Furthermore, the device has a total of 9 GPIO pins and 1 analog pin.

The most important feature for this project is a built-in Wifi antenna (supporting 802.11n) and TCP/IP stack, which permits for the implementation of an “email service” to send updates about plant status without increasing the cost of the device further by adding a clunky Wifi shield onto the list of parts.

All sensors are powered from a Digital I/O pin. This allows the MCU to turn the sensor circuits On/Off to reduce power consumption as these would constantly consume power if connected directly to the battery. The ESP8266 also has an internal 1.1V LDO that its ADC chip uses for voltage reference which allows for battery voltage to be read. Finally, a switch between EN pin and GND allows the user to turn the entire device On/Off in case the device is not in use. When closed, this pulls the EN pin low and disables the 3.3V LDO voltage regulator that powers the entire system.

The sensor components attached to the system are simple resistive devices--a photoresistor, thermistor, and soil moisture sensor. All sensors are configured in a voltage divider circuit to allow for easy resistance calculations. Additionally, a voltage divider circuit is constructed on the battery input and connected to the analog pin of the ESP8266 to allow for battery voltage measurements.

This is particularly important as LiPo-based battery (which is what this project intends to use as a power source) cells will become damaged if they are discharged below a threshold voltage. A logic-level enhancement-mode NMOS is placed in the middle of this voltage divider circuit. Its gate is connected to Digital Pin 4 to allow the MCU to turn the circuit On/Off to turn reduce power consumption when battery readings are unnecessary.

An SPI-based dedicated analog to digital converter chip, MCP3008, handles reading all relevant sensor values. It references the output voltage of Digital Pin 4, which all the sensors are powered from, to make measurements.

The following calculations were obtained for maximum current through each sensor circuit, assuming that the LDR and NTC thermistor have a value of 0Ω (conservative approach as this is unlikely to be the case) and V_{In} of 3.3V:

- MCP3008 ADC I_{max}: 500 µA
- Photoresistor I_{max}: 330 µA
- Thermistor I_{max}: 330 µA
- Moisture sensor I_{max}: 6.346 µA
- Battery voltage sensing circuit I_{max}: 6.346 µA

As the sensors and are connected in parallel to the same GPIO pin (aside from the battery sensing circuit), their total current was calculated by summing the individual currents:

- Total Sensor I_{max}: 1.173 mA

The ESP8266 has a max current draw of 120 mA while transmitting data and a max current draw of 20 µA while in Deep Sleep Mode. Assuming that the device is configured for hourly updates with sensor readings take place in ~1sec, transmission takes place in ~60 sec, and the device entering Deep Sleep Mode for the rest of the hour, an average current draw of 2mA was obtained. With a 2200mAh LiPo battery, the theoretical lifespan of the device (before requiring recharge) was determined to be 32 days[6]. The authors feel that this meets the requirement set forth of a low-power system. Furthermore, the lifespan would greatly increase by decreasing the update frequency.

4. Software Design

The following flowchart provides a general overview of how the entire system is designed to function:

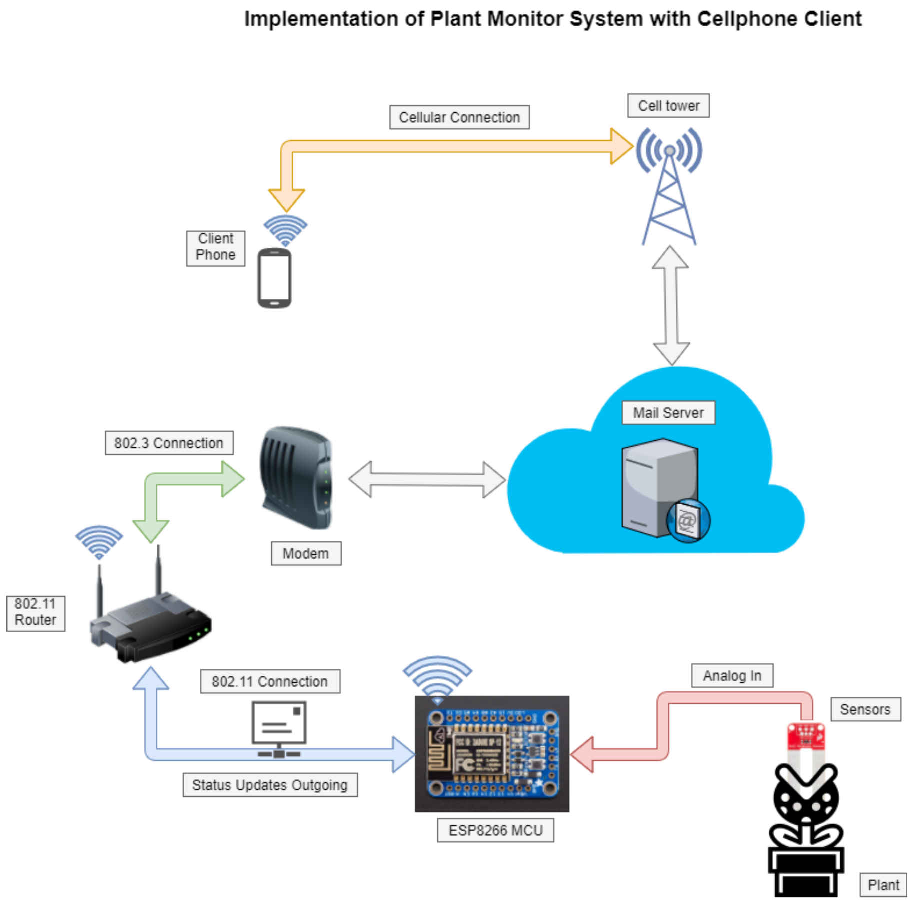


Figure 4.1: Plant Monitor System overview

The software design for the ESP8266 MCU is relatively straightforward; all sensor readings can be measured via Analog-to-Digital Conversion Modules, converted to appropriate units, and compacted into a single array of sensor values. As a result, the actual software logic can be broken down into a few important phases:

- Initialization
- Collect ADC Measurements
- Convert to appropriate units
- Format message
- Connect to WiFi router
- Send Email message
- Configure Deep Sleep With Timeout

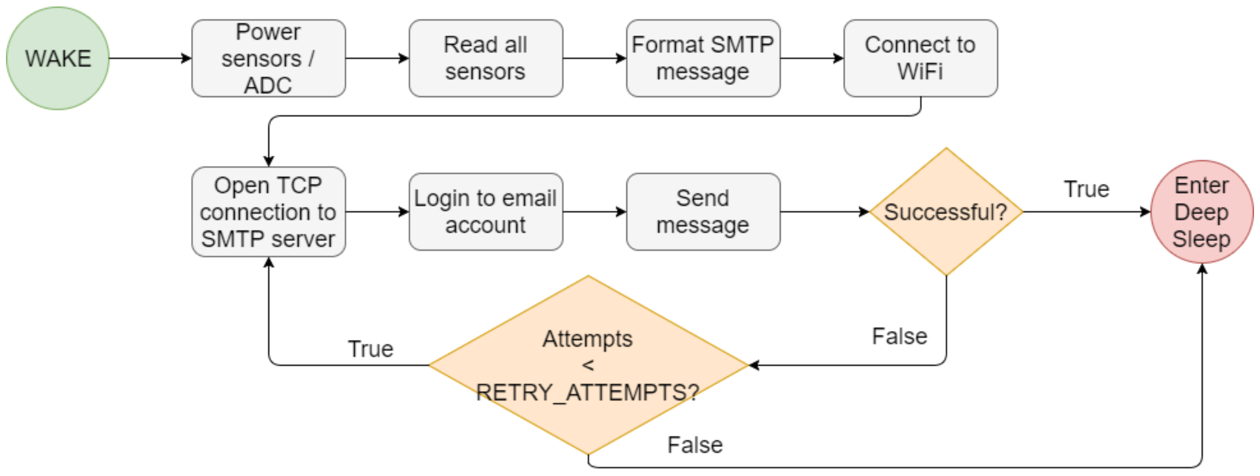


Figure 4.2: ESP8266 software flowchart

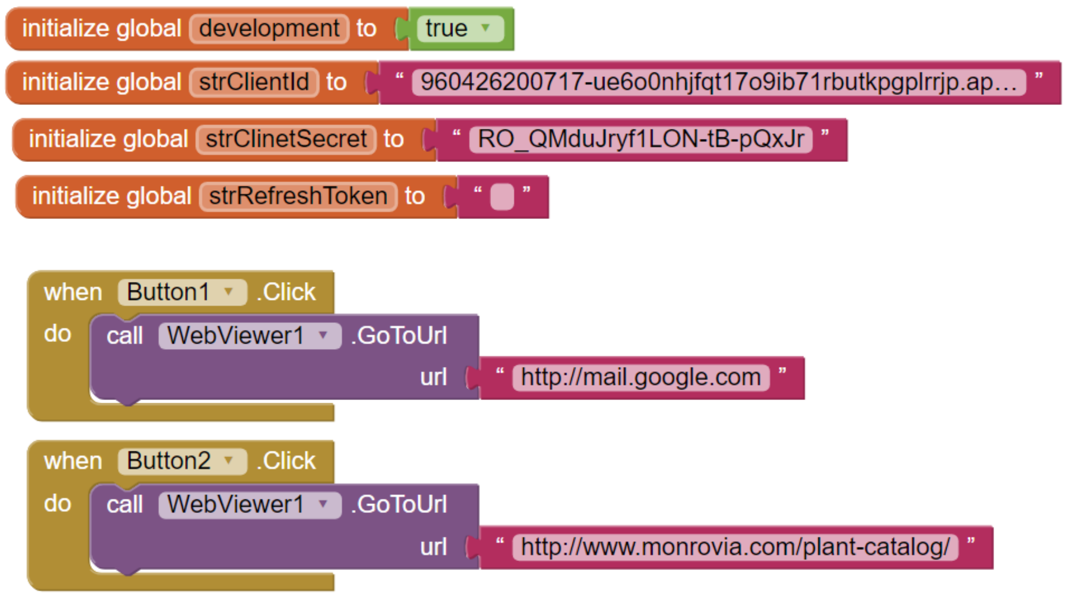
The Initialization phase covers the basic startup tasks; UART for diagnostics purposes is initialized and pins are configured. An astute observer will notice that there isn't any code in the main loop and all phases are contained within the setup function. There is a reason for this--the Deep Sleep functionality for this particular microcontroller completely resets the system, such that it will return back to the initial Setup/Initialization stage starting point when it wakes up. The only way to store variables between Deep Sleep cycles is to either store in Flash memory, EEPROM, or the RTC module's RAM.

Communications with each of the sensors is handled through the separate ADC module; a library has been written for this purpose in order to enhance code readability and encapsulate much of the extra steps in wrapper functions. The unit conversion functions for each of the sensor's ADC measurements are also contained within the SensorMetrics library.

After sensor measurements have been taken, the system waits until a connection has been established with a preprogrammed Wifi network. Once this occurs, it begins attempting to send an email to the specified address and will repeat this operation until it succeeds or reaches the maximum retry attempts. Retry attempts increase the timeout to wait between TCP packets to attempt to resolve communications issues.

The Deep Sleep phase exists as a termination point for the software logic; as mentioned, it resets almost the entire processor when going to sleep and restarts at the Setup phase when waking. A delay is configured using the onboard Realtime Clock and Calendar module, which currently causes the device to wake up once an hour. Due to some engineering flaws with this particular microcontroller, there is some drift involved with doing so (which is elaborated on further in the Conclusions section).

In addition to the ESP8266 software, a simple Android app was developed for testing purposes using MIT's App Inventor. This essentially wraps the SMTP server's mailbox web page and displays it within an app. Additionally, the app wraps a plant reference site to provide the user a convenient method to search for various plant lighting, temperature, and moisture level metrics.

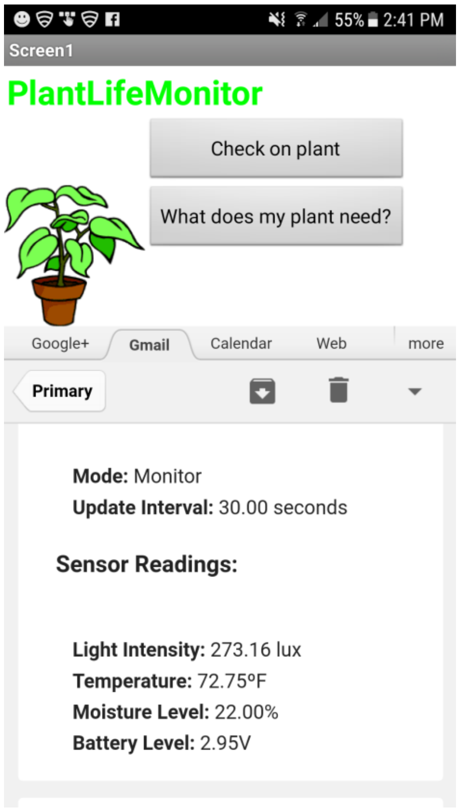


```
initialize global development to true
initialize global strClientId to " 960426200717-ue6o0nhjfqt17o9ib71rbutkpgplrrjp.ap... "
initialize global strClinetSecret to " RO_QMduJryf1LON-tB-pQxJr "
initialize global strRefreshToken to "  "

when Button1 .Click
do  call WebViewer1 .GoToUrl
                       url " http://mail.google.com "

when Button2 .Click
do  call WebViewer1 .GoToUrl
                       url " http://www.monrovia.com/plant-catalog/ "
```

Figure 4.3: Android app block code

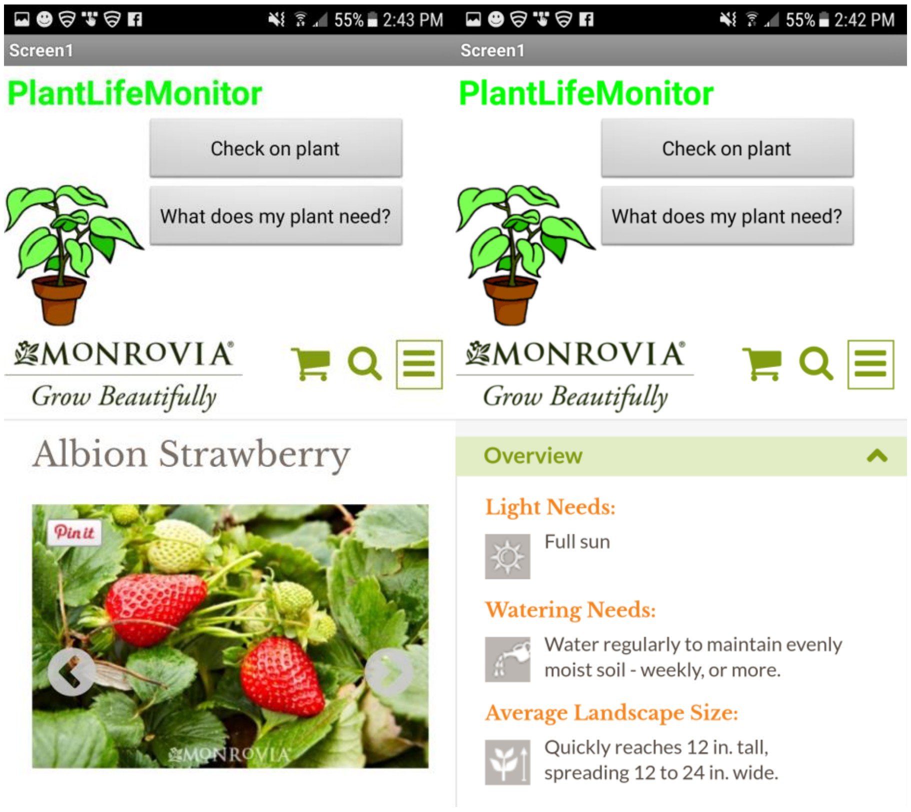


Mode: Monitor
Update Interval: 30.00 seconds

Sensor Readings:

Light Intensity: 273.16 lux
Temperature: 72.75°F
Moisture Level: 22.00%
Battery Level: 2.95V

Figure 4.4: SMTP wrapper page



Light Needs:
Full sun

Watering Needs:
Water regularly to maintain evenly moist soil - weekly, or more.

Average Landscape Size:
Quickly reaches 12 in. tall, spreading 12 to 24 in. wide.

Figure 4.5: Plant reference site wrapper page

5. Source Code

5.1 *FinalProject.ino*

```
/*
 *  Authored by Jeremy Maxey-Vesperman with consultation from Zachary Goldasich
 *  Submitted 12/14/2017
 */

//Include wifi library for ESP8266
#include <ESP8266WiFi.h>
#include "SMTP.h"
#include "SensorMetrics.h"

/* Definitions */
#define RED_LED_PIN 0
#define BLUE_LED_PIN 2
#define CONFIG_UDPATE_PIN 5

#define BAUD_RATE 115200

#define DEFAULT_NETWORK_NAME "PlantLifeMonitor"
#define DEFAULT_NETWORK_PASS "PlzSenpaiGiveMeA"
#define DEFAULT_HOSTNAME "Plant_Node_1"

#define STARTUP_WAIT 5000
#define RETRY_ATTEMPTS 3
#define RETRY_DELAY 5000
#define TIMEOUT_INCREMENT 500
#define SLEEP_TIME 30e6 // 30e6 µSec is 30 seconds

#define SMTP_SUBJECT_LINE "Plant Alert!"

/* Global variables */
/* *** INSERT PLANT MONITOR EMAIL ADDRESS HERE *** */
SMTP updater("PLANT@MONITOR.EMAIL");
int attempts = 1;
bool f_Sent = false;

void setup() {

  // startup delay
  // wait so that a normal human has time to connect the serial monitor before program starts
  delay(STARTUP_WAIT);
  // begin serial communication at the defined baud rate
  Serial.begin(BAUD_RATE);

  // Wait for serial to initialize.
  while(!Serial) { }

  pinMode(0, OUTPUT); // red LED
  pinMode(2, OUTPUT); // blue LED
  digitalWrite(0, HIGH);
  digitalWrite(2, HIGH);

  // Turn on red LED to indicate sensor reading is in progress (remove for actual product
implementation)
  digitalWrite(RED_LED_PIN , LOW);
  float * sensors = SensorMetrics::getSensors(); // read all sensors and store them in a float
array
  digitalWrite(RED_LED_PIN, HIGH); // turn off the red LED (remove for actual product
implementation)
```

```
  // Init variables for creating SMTP message
  String emailSubject = SMTP_SUBJECT_LINE; // Subject line of email message
  String emailBody = "";

  emailBody += "<h2><u>Plant Node 1</u></h2><br>";
  emailBody += "<h3>&emsp;Configuration:</h3><br>";
  emailBody += "&emsp;&emsp;<b>Mode:</b> Monitor<br>";
  emailBody += "&emsp;&emsp;<b>Update Interval:</b> " + String(SLEEP_TIME/1000000) + "
seconds<br>";
  emailBody += "<h3>&emsp;Sensor Readings:</h3><br>";

  for(int i = 0; i < SensorMetrics::_SENSOR_DESCRIPTIONS_SIZE; i++) {
    emailBody += ("<b>" + SensorMetrics::_SENSOR_DESCRIPTIONS[i][0] + "</b>" + sensors[i] +
SensorMetrics::_SENSOR_DESCRIPTIONS[i][1]);
  }

  delete [] sensors; // Clean up memory

  WiFi.forceSleepWake(); // workaround for Deep Sleep Mode bug #2186
  WiFi.hostname(DEFAULT_HOSTNAME); // change hostname to something more friendly
  // attempt to connect to the specified wireless network
  WiFi.begin(DEFAULT_NETWORK_NAME, DEFAULT_NETWORK_PASS);

  Serial.print("Connecting...");

  // Wait to be connected
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();

  // Let the user know we've connected and what our IP address is
  Serial.print("Connected!\tIP Address: ");
  Serial.println(WiFi.localIP());

  // Turn on blue LED to indicate an attempt to email (remove for actual product
implementation)
  digitalWrite(BLUE_LED_PIN, LOW);
  // atempt to send the email up to 3 times before waiting for next cycle
  while(attempts <= RETRY_ATTEMPTS && !f_Sent) {
    Serial.println("\r\nAttempt #" + String(attempts));
    if(updater.sendUpdateEmail(emailSubject, emailBody)) { // if we successfully sent the
update email...
      f_Sent = true; // indicate to the loop that the email sent successfully
    }
    else {
      // increment the timeout by TIMEOUT_INCREMENT per attempt to try to resolve comms issues
      int timeout = updater.getTimeout();
      timeout += TIMEOUT_INCREMENT;
      updater.updateTimeout(timeout);
      attempts++;
      // wait 5 seconds before trying again
      delay(RETRY_DELAY);
    }
  }
  digitalWrite(BLUE_LED_PIN, HIGH); // turn off the blue LED (remove for actual product
implementation)

  Serial.println("Going into deep sleep for " + String(SLEEP_TIME/1000000) + " seconds");
  ESP.deepSleep(SLEEP_TIME, WAKE_RF_DEFAULT);
}

/* main loop */
void loop() {

}
```

```
// ©2017 Jeremy Maxey-Vesperman
```

5.2 *SMTP.cpp*

```
/*
 *  Authored by Jeremy Maxey-Vesperman with consultation from Zachary Goldasich
 *  Submitted 12/14/2017
 */

// Include necessary header files
#include "Arduino.h"
#include "SMTP.h"
#include <ESP8266WiFi.h>

/* Definitions */
// Only support connection to Google's SMTP server through SSL socket
#define SMTP_SERVER "smtp.gmail.com"
#define SMTP_PORT 465
#define DEFAULT_RESPONSE_TIMEOUT 750 // Default wait time for responses from server


// Various SMTP commands and friendly names of commands
#define HELO_CMD "HELO Seedling"
#define AUTH_LOGIN_CMD "AUTH LOGIN"
#define USERNAME_CMD "Username"
#define PASSWORD_CMD "Password"
#define MAIL_FROM_CMD "MAIL FROM: <plantlifemonitor@gmail.com>" // No support for sending from
different address atm
#define RCPT_TO_CMD_1 "RCPT TO: <"
#define RCPT_TO_CMD_2 ">"
#define DATA_CMD "DATA"
/* *** INSERT PLANT MONITOR EMAIL ADDRESS HERE *** */
#define FROM_INPUT "From: Plant Life Monitor <PLANT@MONITOR.EMAIL
#define TO_INPUT "To: "
#define SUBJECT_INPUT "Subject: "
#define EOM_CMD "EOM"
#define QUIT_CMD "QUIT"
#define SMTP_TERMINATE_CHAR "." // EOM character for body of email
// REMOVE THIS AND FORCE USER TO SPECIFY RECIPIENT ADDRESS AT SETUP
// *** INSERT DEFAULT RECIPIENT EMAIL ADDRESS
#define DEFAULT_RECIPIENT_ADDR "DEFAULT@RECIPIENT.EMAIL" // Default recipient address

// Base64 encoded form of username and password for sender mailbox
// *** INSERT BASE 64 ENCODED EMAIL ADDRESS AND PASSWORD FOR PLANT MONITOR EMAIL ACCOUNT
#define B64_USERNAME "INSERT_B64_ENCODED_EMAIL"
#define B64_PASSWORD "INSERT_B64_ENCODED_PASSWORD"

// SMTP server response codes
#define RESP_SERVICE_READY 220
#define RESP_ACTION_OKAY 250
#define RESP_AUTH_LOGIN 334
#define RESP_AUTHENTICATED 235
#define RESP_START_MAIL 354
#define RESP_CLOSE 221

// Formatting for error messages
#define INVAL_RESP_1 "Did not receive response "
#define INVAL_RESP_2 " to "
#define INVAL_RESP_3 " command."

// Diagnostics messages
#define EMAIL_SUCCESS_MSG "Email Sent Successfully!"
#define SMTP_CONNECT_SUCCESS_MSG "Successfully connected to SMTP Server!"
#define SMTP_CONNECT_FAILED_MSG "Failed to connect to SMTP server"
#define HELO_MSG "Issuing HELO command"
#define AUTH_LOGIN_MSG "Issuing AUTH LOGIN command"
#define USERNAME_MSG "Issuing USERNAME command"
```

```
#define PASSWORD_MSG "Issuing PASSWORD command"
#define MAIL_FROM_MSG "Issuing MAIL FROM command"
#define RCPT_TO_MSG "Issuing RCPT TO command: "
#define DATA_MSG "Issuing DATA command"
#define BODY_MSG "Sending body of message..."
#define EOM_MSG "Sending EOM character"
#define QUIT_MSG "Issuing QUIT command"

/* Variables */
WiFiClientSecure _smtpClient; // create a secure client object
int _timeout;
String _recipientAddr;
// Output message and flag
String _emailStatusMsg;
boolean f_EmailSuccessful;

/* Constructors */
// Default constructor uses default recipient address and timeout
SMTP::SMTP()
  : _recipientAddr(DEFAULT_RECIPIENT_ADDR), _timeout(DEFAULT_RESPONSE_TIMEOUT)
{
}

// Constructor that initializes recipient address to the one specified
SMTP::SMTP(String recipientAddr)
  : _recipientAddr(recipientAddr), _timeout(DEFAULT_RESPONSE_TIMEOUT)
{
}

/* Functions */

/* Method for sending update email via SMTP */
bool SMTP::sendUpdateEmail(String subject, String body) {
  // reinit output variables of the instance
  _emailStatusMsg = EMAIL_SUCCESS_MSG;
  f_EmailSuccessful = true;

  // if we successfully connect to the SMTP server...
  if (_smtpClient.connect(SMTP_SERVER, SMTP_PORT)) { // connect via SSL socket
    Serial.println(SMTP_CONNECT_SUCCESS_MSG);

    // create the recipient command string from the specified recipient address
    String recipientCmd = (RCPT_TO_CMD_1 + _recipientAddr + RCPT_TO_CMD_2);

    Serial.println(HELO_MSG);
    sendCmd(HELO_CMD, RESP_SERVICE_READY, HELO_CMD); // issue HELO command
    Serial.println(AUTH_LOGIN_MSG);
    sendCmd(AUTH_LOGIN_CMD, RESP_AUTH_LOGIN, AUTH_LOGIN_CMD); // issue authentication command
    Serial.println(USERNAME_MSG);
    sendCmd(B64_USERNAME, RESP_AUTH_LOGIN, USERNAME_CMD); // send username
    Serial.println(PASSWORD_MSG);
    sendCmd(B64_PASSWORD, RESP_AUTHENTICATED, PASSWORD_CMD); // send password
    Serial.println(MAIL_FROM_MSG);
    sendCmd(MAIL_FROM_CMD, RESP_ACTION_OKAY, MAIL_FROM_CMD); // specify address we are sending
from
    Serial.println(RCPT_TO_MSG + recipientCmd);
    sendCmd(recipientCmd, RESP_ACTION_OKAY, recipientCmd); // specify address of the receiver
    Serial.println(DATA_MSG);
    sendCmd(DATA_CMD, RESP_START_MAIL, DATA_CMD); // issue DATA command to start sending
message

    Serial.println(BODY_MSG);

    _smtpClient.println(FROM_INPUT); // send the From line
    delay(_timeout);
    _smtpClient.println(SUBJECT_INPUT + subject); // send the Subject line
    delay(_timeout);
    _smtpClient.println(TO_INPUT + _recipientAddr); // send the To line
      delay(_timeout);
    _smtpClient.println("Mime-Version: 1.0;\r\nContent-Type: text/html; charset=\"ISO-8859-
1\r\nContent-Transfer-Encoding: 7bit;");
    delay(_timeout);
      _smtpClient.println();
    delay(_timeout);
    _smtpClient.println("<html>\r\n<body>\r\n" + body + "\r\n</body></html>"); // send message
    delay(_timeout);

    Serial.println(EOM_MSG);
    sendCmd(SMTP_TERMINATE_CHAR, RESP_ACTION_OKAY, EOM_CMD); // send terminating character to
end message
    Serial.println(QUIT_MSG);
    sendCmd(QUIT_CMD, RESP_CLOSE, QUIT_CMD); // issue QUIT command to terminate session

    // output status message upon completion and return flag
    Serial.println(_emailStatusMsg);

    _smtpClient.stop(); //

    return f_EmailSuccessful;
  }
  else { // otherwise... output error and return false
    Serial.println(SMTP_CONNECT_FAILED_MSG);
    return false;
  }
}

/* Getter method to return timeout for connecting to SMTP server */
int SMTP::getTimeout() {
  return _timeout;
}

/* Setter method to update recipient email address */
// SHOULD ADD SOME WAY TO VERIFY VALID EMAIL FORMAT
void SMTP::updateRecipientAddr(String newAddr) {
  _recipientAddr = newAddr;
}

/* Setter method to update timeout for connecting to SMTP server */
void SMTP::updateTimeout(int newTimeout) {
  if(newTimeout > 0) { // if new timeout is a valid timeout setting...
    _timeout = newTimeout; // update
  }
  else { // otherwise...
    // allows for reseting to default without knowing the default setting
    _timeout = DEFAULT_RESPONSE_TIMEOUT; // reset to default
  }

}

/* Issues SMTP command and checks the response if sending email hasn't already failed */
void SMTP::sendCmd(String command, int expectedResponse, String cmdFriendlyName) {
  if (f_EmailSuccessful) { // if we haven't failed yet...
    _smtpClient.println(command); // issue the specified command
    if (readResponse() != expectedResponse) { // if the response code is not the expected
one...
        _emailStatusMsg = genInvalResponse(cmdFriendlyName, expectedResponse); // update status
message
        f_EmailSuccessful = false; // set the success flag to false
      }
  }
}

/* Reads entire response returned by the server. Returns the response code to the caller. */
int SMTP::readResponse() {
  String response = "";
  int i = 0;

  delay(_timeout); // wait specified amount of time for a response

  while(_smtpClient.available()) { // while there is something to read...
```

```
      char c = _smtpClient.read(); // read the character...
      Serial.print(c); // output the character...

      if(i < 3) { // first 3 characters are the first response code
        response += c;
        i++;
      }
    }

  if (response == "") { // return -1 if response not received before timeout
    return -1;
  }

  // else... return response code converted to int or 0 if no valid response string in first 3
characters
  return response.toInt();
}

/* Generates the invalid response string from response code expected and the command sent */
String SMTP::genInvalResponse(String command, int responseCode) {
  return (INVAL_RESP_1 + String(responseCode) + INVAL_RESP_2 + command + INVAL_RESP_3);
}

// ©2017 Jeremy Maxey-Vesperman
```

5.3 *SensorMetrics.cpp*

```
/*
 *  Authored by Jeremy Maxey-Vesperman with consultation from Zachary Goldasich
 *  Submitted 12/14/2017
 */

// Include necessary header files
#include "Arduino.h"
#include "SensorMetrics.h"
#include <SPI.h>
#include <math.h>

/* Definitions */

// Delays
#define SENSOR_STABILIZE_DELAY 500 // ms

// ADC reference info
#define ADC_RESOLUTION 1024 // both adcs are 10-bit resolution
#define EXTERN_ADC_V 2.84 // MCP3008 is setup to reference voltage from DIO pin on ESP8266
#define INTERN_ADC_V 1.1 // ESP8266 internal adc has 1.1V LDO for voltage referencing

// External ADC (MCP3008)
#define EXTERN_ADC_SPI_FREQ 18000 // clock frequency for SPI comms with external adc
#define SENSOR_EN_PIN 4 // pin for powering the external adc chip and sensors
#define SS_PIN 15 // Slave Select pin for external adc
#define PHOTORESISTOR_CH 1 // channel of external adc that photoresistor is connected to
#define THERMISTOR_CH 2 // channel of external adc that thermistor is connected to
#define MOISTURE_SENSOR_CH 4 // channel of external adc that moisture sensor is connected to

// Photoresistor info (WDYJ GM5539)
#define PHOTORESISTOR_R2 9830 // photoresistor voltage divider circuit R2 value
// Best fit line equation values generated from test measurements
#define PHOTORESISTOR_M -1.1116
#define PHOTORESISTOR_B 7.3113

// Thermistor info (Honeywell 192-103LET-A01)
#define THERMISTOR_R2 9930 // thermistor voltage divider circuit R2 value
#define THERMISTOR_BETA 3974 // Beta value of thermistor ()
#define THERMISTOR_R0 10000
#define THERMISTOR_T0 298.15
```

```
// Moisture sensor info (XCSOURCE TE215)
#define MOISTURE_SENSOR_R1 406100 // moisture sensor voltage divider circuit R1 value
#define MOISTURE_SENSOR_R2 129900 // moisture sensor voltage divider circuit R2 value
#define MOISTURE_MIN 2800 // Output (mV) at 0% moisture
#define MOISTURE_MAX 500 // Output (mV) at 100% moisture

// Battery voltage divider circuit info
#define BATTERY_R1 389700
#define BATTERY_R2 128100

// Constants for temperature conversions and calculations
#define ROOM_TEMP_K 298.15
#define TEMP_F_MULT 1.8
#define TEMP_F_CONST 459.67
#define TEMP_C_CONST 273.15

/* Constants */

const int SensorMetrics::_SENSOR_DESCRIPTIONS_SIZE = 4;
const String SensorMetrics::_SENSOR_DESCRIPTIONS[_SENSOR_DESCRIPTIONS_SIZE][2] = {
{"&emsp;&emsp;Light Intensity: ", " lux<br>"},

{"&emsp;&emsp;Temperature: ", "°F<br>"},

{"&emsp;&emsp;Moisture Level: ", "%<br>"},

{"&emsp;&emsp;Battery Level: ", "V<br>"}};

/* Functions */

// Function for turning on the external ADC and sensors
void SensorMetrics::externADCOn() {
    digitalWrite(SS_PIN, HIGH); // deselect the ADC (needs a falling edge)
    digitalWrite(SENSOR_EN_PIN, HIGH); // turn on the ADC and sensors
    delay(SENSOR_STABILIZE_DELAY); // give time for adc to turn on and sensor voltages to
stabilize
}

// Function for turning off the external ADC and sensors
void SensorMetrics::externADCOff() {
    digitalWrite(SS_PIN, LOW); // save energy by turning off the pin
    digitalWrite(SENSOR_EN_PIN, LOW); // turn off the ADC and sensors
}

// Function for setting up pins and SPI for external ADC and sensors
void SensorMetrics::setupExternADC() {
    pinMode(SENSOR_EN_PIN, OUTPUT);
    pinMode(SS_PIN, OUTPUT);

    SPI.begin();
    SPI.setBitOrder(MSBFIRST); // MSB bit sent first
    SPI.setDataMode(SPI_MODE0);
    SPI.setFrequency(EXTERN_ADC_SPI_FREQ); // 18kHz / 18 = 1ksa/s (1ms readings)
}

// Function for getting readings from the ESP8266's internal ADC
// ADC has an internal voltage reference of 1.1V
int SensorMetrics::readInternADC() {
  int reading = analogRead(A0); // read from A0 (ADC pin)

  return reading; // return the reading
}

// Function for reading from external SPI ADC chip
// ADC has voltage reference of digital pin output voltage on ESP8266
int SensorMetrics::readExternADC(int ch) {
  if (ch > 7 || ch < 0) { // MCP3008 has 8 channels
    return -1; // return -1 if passed invalid channel argument
  }
  digitalWrite(SS_PIN, LOW); // select ADC
  SPI.transfer(B00000001); // send 7 leading 0's and the START bit
  // send control bits (single-ended + channel address shifted into upper 4 bits)
  uint8_t reading_upper = SPI.transfer((ch + 8) << 4); // store null bit + upper 2 bits of adc
reading
  uint8_t reading_lower = SPI.transfer(0); // store lower byte of adc reading

  digitalWrite(SS_PIN, HIGH); // deselect ADC

  // clear null bit, shift first byte of reading into upper byte of int, bitwise OR second byte
with lower byte of int
  return (((reading_upper & 3) << 8) + reading_lower); // return 10-bit ADC reading for
specified channel
}

// Function to convert adc value into voltage measurement
float SensorMetrics::getVMeas(int adc, bool externADC) {
  // Compute first part of vMeas by getting adc reading ratio
  float vMeas = ((float(adc) + 1.0) / float(ADC_RESOLUTION)); // resolution is same between
internal and external chip

  // If we are reading from external adc...
  if (externADC) {
    vMeas *= EXTERN_ADC_V; // multiply by external adc reference voltage
  }
  // Otherwise...
  else {
    vMeas *= INTERN_ADC_V; // multiply by internal adc reference voltage
  }

  return vMeas; // return the voltage measured
}

// Function to calculate input voltage to voltage divider circuit
float SensorMetrics::getVDivVin(float r1, float r2, float vDrop, bool vDropAcrossR1) {
  // Calculate input voltage to divider circuit (AKA battery voltage) based on known R1 and R2
  float vIn = (vDrop * (r1 + r2));

  // Divide by appropriate resistor value
  if (vDropAcrossR1) {
    vIn /= r1;
  }
  else {
    vIn /= r2;
  }

  return vIn; // return the input voltage of the voltage divider
}

// Function to convert measured voltage into R1 value of a voltage divider circuit
float SensorMetrics::getVDivR1(float vMeas, int r2, bool externADC) {
  float r1;

  // Use appropriate reference voltage based on which adc took the reading...
  if (externADC) {
    r1 = ((EXTERN_ADC_V * r2) - (vMeas * r2)) / vMeas;
  }
  else {
    r1 = ((INTERN_ADC_V * r2) - (vMeas * r2)) / vMeas;
  }

  return r1; // return the value of r1 in Ohms (Ω)
}

// Function to get light intensity reading (Lux) from photoresistor
float SensorMetrics::getLux (bool externADC) {
  int prADC;
  float lux, vMeas, photoResistor;
  // Read the adc channel that the photoresistor is attached to
  if (externADC) {
    prADC = readExternADC(PHOTORESISTOR_CH);
  }
  else {
    prADC = readInternADC();
  }

  vMeas = getVMeas(prADC, externADC); // convert adc reading to voltage
  // Get value of photoresistor based on vMeas and the value of the known R2 resistor in the
voltage divider circuit
  photoResistor = getVDivR1(vMeas, PHOTORESISTOR_R2, externADC);

  // Get light intensity in lux using photoresistor value and variables from line of best fit
for the sensor
  lux = pow(photoResistor, PHOTORESISTOR_M) * pow(10, PHOTORESISTOR_B); // *** m and b are
unique to the photoresistor

  return lux; // return the light intensity value in lux
}

// Function to get temperature in Kelvin from thermistor
float SensorMetrics::getTempK (bool externADC) {
  int thADC;
  float tempK, vMeas, thermistor;

  // Read the adc channel that the thermistor is attached to
  if (externADC) {
    thADC = readExternADC(THERMISTOR_CH);
  }
  else {
    thADC = readInternADC();
  }

  vMeas = getVMeas(thADC, externADC); // convert adc reading to voltage
  // Get value of thermistor based on vMeas and the value of the known R2 resistor in the
voltage divider circuit
  thermistor = getVDivR1(vMeas, THERMISTOR_R2, externADC);

  // Return the room temp in Kelvin constant if thermistor value = the thermistor r0 value
(resistance of thermistor @ room temp in K)
  // This prevents divide by 0 error in conversion formula in the event that reading is exactly
room temperature (ln(1) = 0...)
  if (thermistor == THERMISTOR_R0) {
    tempK = ROOM_TEMP_K;
  }
  // Otherwise... use the conversion formula to get temp in K from the resistance value of
thermistor
  else {
    tempK = (((THERMISTOR_T0 * THERMISTOR_BETA) / (log(THERMISTOR_R0 / thermistor))) /
(((THERMISTOR_BETA) / (log(THERMISTOR_R0 / thermistor))) - THERMISTOR_T0));
  }

  return tempK; // return the temperature reading in Kelvin
}

// Function to get temperature in Fahrenheit from thermistor
float SensorMetrics::getTempF (bool externADC) {
  float tempK, tempF;

  tempK = getTempK(externADC); // get the temperature in Kelvin
  tempF = ((tempK * TEMP_F_MULT) - TEMP_F_CONST); // convert to Fahrenheit from Kelvin

  return tempF; // return the temperature reading in Fahrenheit
}

// Function to get temperature in Celsius from thermistor
float SensorMetrics::getTempC (bool externADC) {
  float tempK, tempC;
  tempK = getTempK(externADC); // get the temperature in Kelvin
  tempC = (tempK + TEMP_C_CONST); // convert to Celsius from Kelvin

  return tempC; // return the temperature reading in Celsius
}

// Function to get moisture level %
float SensorMetrics::getMoistureLvl(bool externADC) {
  int moistADC;
  float moistLvl, vMoist, vMeas;

  moistADC = readExternADC(MOISTURE_SENSOR_CH); // get reading from external adc
  vMeas = getVMeas(moistADC, externADC); // convert to voltage using appropriate reference 
voltage
  vMoist = ((getVDivVin(MOISTURE_SENSOR_R1, MOISTURE_SENSOR_R2, vMeas)) * 1000); // calculate 
input voltage (mV) divider circuit with R1 drop
  moistLvl = map(vMoist, MOISTURE_MIN, MOISTURE_MAX, 0, 100); // map millivolts to %

  return moistLvl; // return moisture % reading
}

// Function to get the battery voltage
float SensorMetrics::getBatteryLvl() {
  int batADC;
  float vBat, vMeas;

  batADC = readInternADC(); // get reading from internal adc
  vMeas = getVMeas(batADC, false); // convert to voltage using appropriate reference voltage
  vBat = getVDivVin(BATTERY_R1, BATTERY_R2, vMeas); // calculate input voltage to divider 
circuit with R2 drop

  return vBat; // return the battery voltage reading
}

float* SensorMetrics::getSensors() {
    float lux, temp, moisture, battery;

    setupExternADC(); // setup external ADC
    externADCOn(); // turn on the ADC and sensors

    lux = getLux(); // get light intensity reading
    temp = getTempF(); // get temperature reading
    moisture = getMoistureLvl(); // get moisture level reading
    battery = getBatteryLvl(); // get battery level reading

    // Create array of sensor values
    float * sensors = new float[4];

    sensors[0] = lux;
    sensors[1] = temp;
    sensors[2] = moisture;
    sensors[3] = battery;

    externADCOff(); // turn off the ADC and sensors

    return sensors; // return array of sensor values
}

// ©2017 Jeremy Maxey-Vesperman
```

5.4 *SMTP.h*

```
/*
 *  Authored by Jeremy Maxey-Vesperman with consultation from Zachary Goldasich
 *  Submitted 12/14/2017
 */
```

```
#ifndef SMTP_h
#define SMTP_h

// Include the necessary libraries
#include "Arduino.h"
#include <ESP8266WiFi.h>

/* SMTP class definition */
class SMTP {
	public:
		/* Constructors */
		SMTP(); // default constructor inits to default timeout and recipient address
		SMTP(String recipientAddr); // constructor that inits to different recipient
address

		/* Public Functions and Methods */
		bool sendUpdateEmail(String subject, String body); // function for sending
update email
		int getTimeout(); // function for getting the current timeout setting
		void updateRecipientAddr(String newAddr); // method for updating the recipient
address
		void updateTimeout(int newTimeout); // method for updating the timeout
	private:
		/* Private Instance Variables */
		WiFiClientSecure _smtpClient; // secure TCP client object
		int _timeout; // how long to wait for a server response
		String _recipientAddr; // email address of the recipient
		String _emailStatusMsg; // output status message
		bool f_EmailSuccessful; // output status flag

		/* Private Functions and Methods */
		void sendCmd(String command, int expectedResponse, String cmdFriendlyName); //
method for sending commands and validating response codes
		int readResponse(); // function for reading server responses
		String genInvalResponse(String command, int responseCode); // function for
generating an invalid server response error message
};

#endif

// ©2017 Jeremy Maxey-Vesperman
```

5.5 *SensorMetrics.h*

```
/*
 *  Authored by Jeremy Maxey-Vesperman with consultation from Zachary Goldasich
 *  Submitted 12/14/2017
 */

#ifndef SensorMetrics_h
#define SensorMetrics_h

// Include the necessary libraries
#include "Arduino.h"
#include <SPI.h>
#include <math.h>

/* SensorMetrics class definition */
class SensorMetrics {
    public:
        /* Public Functions and Methods */
        static void externADCOn();
        static void externADCOff();
        static void setupExternADC();
        static float* getSensors(); // function to get all sensor readings
        static float getLux (bool externADC=true); // function to get light intensity reading
(Lux) from photoresistor
        static float getTempK (bool externADC=true); // function to get temperature in Kelvin
from thermistor
        static float getTempF (bool externADC=true); // function to get temperature in
Fahrenheit from thermistor
        static float getTempC (bool externADC=true); // function to get temperature in Celsius
from thermistor
        static float getMoistureLvl(bool externADC=true); // function to get moisture level %
        static float getBatteryLvl(); // function to get the battery voltage

        /* Public Class Constants */
    static const int _SENSOR_DESCRIPTIONS_SIZE;
        static const String _SENSOR_DESCRIPTIONS[][2];
    private:
        /* Private Functions and Methods */
        static int readInternADC(); // function for getting readings from the ESP8266's
internal ADC
        static int readExternADC(int ch); // function for reading from external SPI ADC chip
        static float getVMeas(int adc, bool externADC=true); // function to convert adc value
into voltage
        static float getVDivVin(float r1, float r2, float vDrop, bool vDropAcrossR1=false); //
function to calculate input voltage to voltage divider circuit
        static float getVDivR1(float vMeas, int r2, bool externADC); // function to convert
measured voltage into R1 value of a voltage divider circuit
};

#endif

// ©2017 Jeremy Maxey-Vesperman
```

6. Conclusions

Overall, the project was a success; though we did not implement as many of our stretch goals due to time constraints and other projects, the core features of the project are not only functioning but also relatively bug-free. The primary goal was to create an inexpensive plant life monitor that was capable of maintaining relatively long battery life via power saving techniques. This has largely been accomplished; it would be a relatively simple matter to integrate additional features on top of the existing logic, as the current program logic is largely a sequence of tasks repeated between every deep sleep cycle.

As far as project difficulties go, the primary issue with this project resided with the Realtime Clock & Calendar module. Though the microcontroller we elected to use does in fact have this functionality, we discovered later that the module has serious drifting issues when in deep sleep mode. In a project such as this, the drift is fairly inconsequential toward the point of the device; however, in a production model it would be necessary to source another microcontroller capable of maintaining the precise time intervals necessary for the application.

7. References

[1] L. Ada, ‘Measuring Light’, 2012. [Online]. Available: https://learn.adafruit.com/photocells/measuring-light

[2] WODEYIJIA TECHNOLOGY CO.,LTD, ‘GM55 Series Datasheet’. [PDF]. Available: http://selfbuilt.net/datasheets/GM55.pdf

[3] D. Williams, ‘Design a Luxmeter Using a Light Dependent Resistor’, 2015. [Online]. Available: https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/

[4] J. Corletto, ‘Measuring Temperature with an NTC Thermistor’, 2016. [Online]. Available: https://www.allaboutcircuits.com/projects/measuring-temperature-with-an-ntc-thermistor/

[5] R. Grisso, M. Alley, D. Holshouser, et. all, ‘Precision Farming Tools: Soil Electrical Conductivity’, 2009. [PDF]. Available: https://pubs.ext.vt.edu/content/dam/pubs_ext_vt_edu/442/442-508/442-508_pdf.pdf

[6] ‘Battery Life Calculator’, 2017. [Online]. Available: https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-battery-life

8. Additional Project References

http://arduino-esp8266.readthedocs.io/en/latest/index.html

http://ai2.appinventor.mit.edu/#5442739243843584

https://www.digikey.com/schemeit/

https://www.losant.com/blog/making-the-esp8266-low-powered-with-deep-sleep

http://www.esp8266.com/viewtopic.php?f=32&t=12722

https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah-esp8266-breakout.pdf

https://sensing.honeywell.com/192-103LET-A01-thermistors

https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf

http://www.seattlerobotics.org/encoder/jun97/basics.html

http://emant.com/316002.page

http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

http://www.esp8266.com/viewtopic.php?f=32&t=12722