Home Automation and Security

Table of Contents

[Hardware 3](#_Toc470216483)

[Breadboard 3](#_Toc470216484)

[Compute 3](#_Toc470216485)

[Network 3](#_Toc470216486)

[Sensors 3](#_Toc470216487)

[Temperature 3](#_Toc470216488)

[Motion 3](#_Toc470216489)

[Light 3](#_Toc470216490)

[Distance 3](#_Toc470216491)

[LED 3](#_Toc470216492)

[Setup 4](#_Toc470216493)

[Software 4](#_Toc470216494)

[Arduino 4](#_Toc470216495)

[Configuration of Arduino Sketch 5](#_Toc470216496)

[ESP8266 5](#_Toc470216497)

[Configuration of ESP8266 5](#_Toc470216498)

[Appendix A: Breadboard hack 6](#_Toc470216499)

[Appendix B: Arduino Sketch 8](#_Toc470216500)

This document describes the configuration, setup and use of a home automation system for a DIY project.

# Hardware

## Breadboard

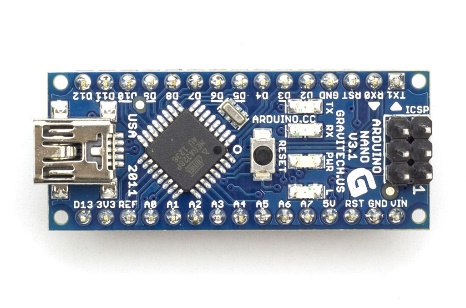
The hardware base is a 400-tie point breadboard that has been hacked as described in Appendix A.

Figure Arduino Nano

## Compute

We will use an Arduino Nano our core compute component (<https://www.arduino.cc/en/Main/ArduinoBoardNano>). This board is currently obsolete but it will work for our purposes.

## Network

An ESP8266 board is used to provide WiFi capability to this setup. The Arduino talks to the ESP8266 using a serial port set up on Pins D2 and D3 of the Arduino. Since the ESP8266 uses 3.3V signals we also need to use a logic level converter to convert from the 5V levels on the Arduino to the 3.3V on the ESP8266.

## Sensors

The following sensors will be used in the system.

### Temperature

DS18B20 sensor in 1 wire mode. This is directly plugged into a digital pin of the Arduino and the temperature is read in units of Centigrade. Any variant of this can be used (e.g. <https://www.adafruit.com/product/381>)

### Motion

Any commodity motion sensor (e.g. <https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/overview>) can be used. All that is required is a digital out that goes high when motion is detected. This is plugged into a digital pin of the Arduino.

### Image result for ldr arduino circuitLight

A commodity light dependent resistor (<https://en.wikipedia.org/wiki/Photoresistor>) is used to detect the light level in the environment. This can be used to sense daylight or if the room lights or on or off. This is connected to an analog pin of the Arduino and reads off the reference voltage. A sample circuit is shown in Figure.

Figure LDR Circuit

### Distance

An ultrasonic sensor is used to measure distance to any object. This could be used to detect a person walking towards or away from the sensor, or to detect motion of a door or car. The HC-SR04 (<https://www.sparkfun.com/products/13959>) sensor is used for this purpose: this has separate Echo and Trigger pins.

### LED

While not a sensor, a high intensity LED is also a part of the system and is used to provide self-contained lighting, e.g. in a hallway or on the stairs. This cannot be driven efficiently using the current from an Arduino pin so a transistor-driven circuit is used to drive the LED. An example LED is <https://www.superbrightleds.com/moreinfo/high-powered/vollong-5w-white-high-power-leds/898/> . Any NPN transistor is used to drive the LED. *I have not tested the reliability of the circuit so YMMV.*

# Setup

Figure 3 shows a picture of the final setup. In the picture the following components are seen:

* Arduino Nano
* 3.3V to 5V level converter
* Carrier socket for the ESP8266 board
* Two micro USB connectors with only the power pins connected. The USB connector on the left powers the Arduino, the level converter and the ESP8266. The one of the right powers the high-intensity LED since that requires additional current.

# Software

Figure Final connections on board

This sections describes the software programming and configuration for all the components used in the system.

## Arduino

The Arduino is the core compute element in this setup. It reads the sensor values at a programmed interval and writes the data to the serial port defined using Pins D2 and D3.

The sensors are mapped to the Arduino pins as shown in the following table.

Table Arduino Pin mapping

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pin No.** | **Sensor Name/Type** | **Direction** | **Data Type** | **Data description** |
| A0 | LDR (Light) | Input | Integer | Ambient light level |
| D2 | Serial port RX | Input | Byte | Data in (to ESP8266) |
| D3 | Serial port TX | Output | Byte | Data out (from ESP8266) |
| D4 | Motion (PIR) | Input | Boolean | HIGH when motion detected |
| D5 | Distance TRIG (Ultrasonic) | Output | Boolean | HIGH to trigger ultrasonic pulse |
| D6 | Distance ECHO (Ultrasonic) | Input | Boolean | HIGH when echo received from object |
| D10 | Temperature | Input | Float | Temperature in Centigrade |
| D12 | LED (Heartbeat) | Output | Boolean | HIGH to turn LED on |
| D13 | LED (High Intensity) | Output | Boolean | HIGH to turn LED on |

The Arduino sketch that collects the sensor data and sends it to the Serial port is given in Appendix B.

### Configuration of Arduino Sketch

Open the Sketch in the Arduino IDE.

Verify the pin numbers match the sensors connected to the Arduino.

Modify the IP Address of the server that the Arduino is going to send the data to. Usually this is a server that has a TCP connection open on the port specified here. This can be a node-red flow or any other TCP listener. This is not required when using the MQTT version.

Modify the SENSORID constant in the sketch. This should be a unique id and is used to identify the sensor output in the logs.

Upload the sketch to the Arduino.

## ESP8266

January 15, 2017 is when I decided to try an MQTT library using Arduino sketch on ESP8266 and it seems to work so far. On the ESP8266 the client name should be unique across the network. The MQTT server IP and port should also be configured correctly.

After multiple days of effort my conclusion is that the traditional AT command set firmware is the most reliable.

ESP-Link was a very attractive proposition but after the initial euphoria stability became a huge issue and I had to give it up.

Attempts to use MQTT firmware also did not help.

As things stand the Arduino-ESP8266 combo is being used for reading the sensor values. Any actions are done locally in the Arduino code. The ESP8266 will forward all action commands received on a MQTT topic to the Arduino.

### Configuration of ESP8266 (not used. Only here for archival purposes. Will be cleaned later)

January 15, 2017: upload the sketch shown in Appendix C to the ESP8266. Verify the following before the upload: ClientName should be unique. MQTT server and port should be correct.

Flash ESP8266 with firmware "v0.9.2.2 AT Firmware.bin" from E:\VenkatDownloads\ESP8266\_flasher\_V00170901\_00\_Cloud Update Ready folder. Use the exe to flash

Login using Serial Monitor and verify baud rate set to 9600. Follow steps in http://electronut.in/an-iot-project-with-esp8266/

Run the following commands from the serial monitor to configure the ESP8266.

* AT (response should be “OK”)
* AT+GMR (response should be firmware version)
* AT+CWMODE? (response should be operation mode. Need to be 1 to avoid multiple SSIDs)
* AT+CWMODE=1 (set operation mode to 1)
* AT+CWLAP (response should be all networks detected)
* AT+CWJAP="ssid","password" (set to network and password)
* AT+CIFSR (verify that we got a password)

For original AI-Thinker firmware use this command to set the baud rate

* AT+UART\_DEF=19200,8,1,0,0

For other firmwares use this command

* AT+CIOBAUD=9600

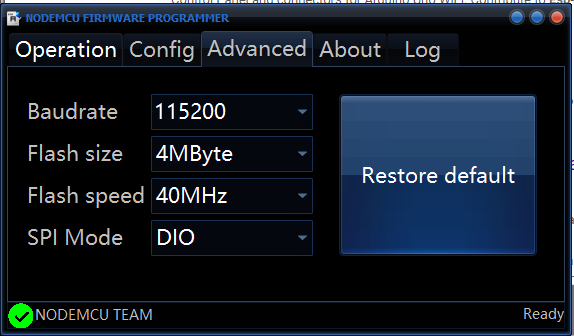
### Configuration of ESP8266 (Unstable. Not used)

Flash ESP8266 with the sketch from Appendix C using the Arduino IDE.

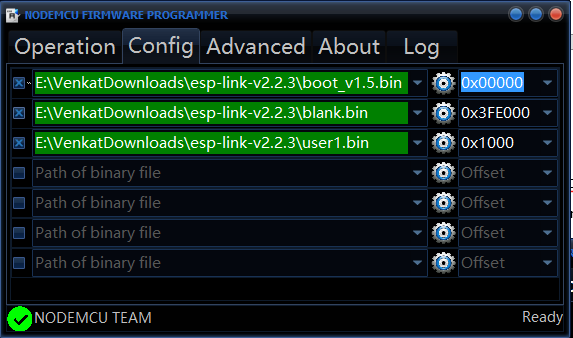
### Configuration of ESP8266 Using esp-link (Unstable. Not used.)

Use the NodeMCU flasher to flash the firmware.

The config screen should look like the following:



Load up the files and offsets as shown here.



# Appendix A: Breadboard hack

For once I am using a title for this post that's as literal as it gets, no euphemisms, no catchy headlines, no provocative calls. This is intended for DIY electronic hobbyists to get their money's worth out of their breadboards.

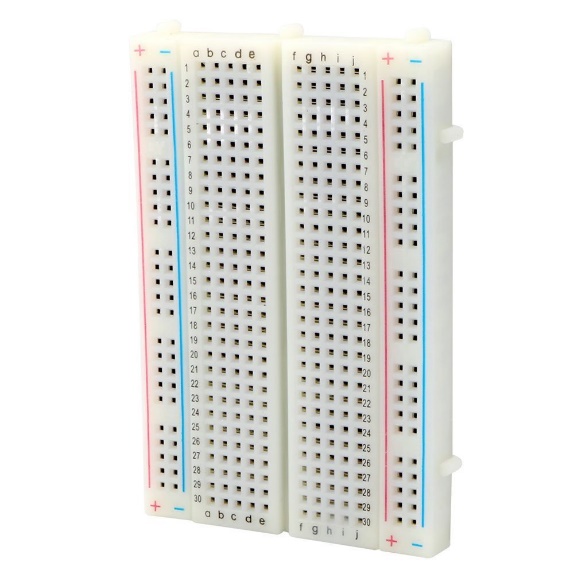
And just in case you are not familiar with the concepts, a [breadboard](http://en.wikipedia.org/wiki/Breadboard) is a hobbyist platform for prototyping electronic circuits. It takes away the need for messy soldering and allows for easy connections between components while you are still building the circuits. In the course of building out my home smarter and safer I've been trying out different stuff and have been using breadboards to install my sensors and interface them to the [Arduino](http://www.arduino.cc/) and [Raspberry Pi](http://www.raspberrypi.org/) platforms.

Figure 400-tie point breadboard

These breadboards come in many different sizes. The smallest I have been using are [these ones](http://www.adafruit.com/product/65) with 18 rows. While they are sufficient for simple circuits like connecting up a few sensors to an Arduino you run out of space very quickly.

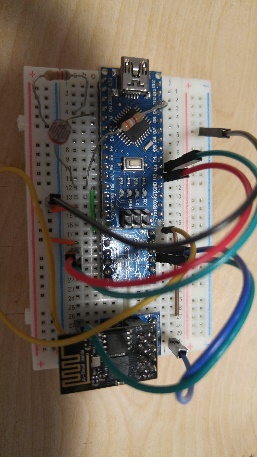
So the next upgrade was to a [400-hole breadboard](http://www.amazon.com/Breadboard-Tie-points-Solderless-Circuit-Arduino/dp/B01GWLVA72/ref=sr_1_7?ie=UTF8&qid=1482431845&sr=8-7&keywords=Small+Bread+Board) which looks something like in the image. This was quite useful since it had space to place an Arduino and access all the pins of the Arduino. In the diagram you see how the +ve and -ve lines are marked with blue and red colors. What this means is that each of the holes ("tie points") along each of the red and blue lines are connected. The idea is to be able to provide multiple voltage lines so that your components on the breadboard can be powered without requiring additional wiring all over the place. However this means that you cannot use those tie points at all.

Figure Cramped breadboard

When you want to connect an Arduino, an ESP8266 and a logic level converter your breadboard looks like this. As you can see, you run out of space pretty quickly.

Hence the idea of hacking the breadboard to create more disconnected tie points so you can add more components. The following images show how you can peel back the adhesive layer behind the breadboard and snip out some of the connections so that each row of 5 tie-points on the power lines become their own row. Essentially you get an additional 19 rows for "free".

So let's begin with the process and see how it goes.

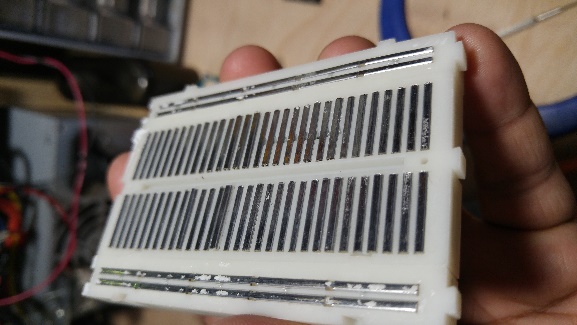


Figure After peeling off adhesive

The first step is to peel away the adhesive layer behind the breadboard and expose the underlying connections between the tie-points. This also gives you an idea of how the tie-points are connected.

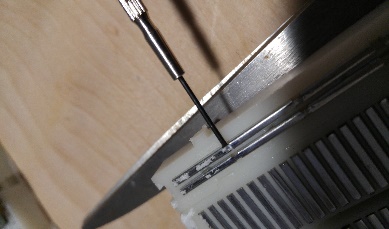


Figure Wedge out strip partially

Use a small screwdriver or pointed tool to gently wedge out the metal strip on the long edge. Be careful not to pull it out all the way.

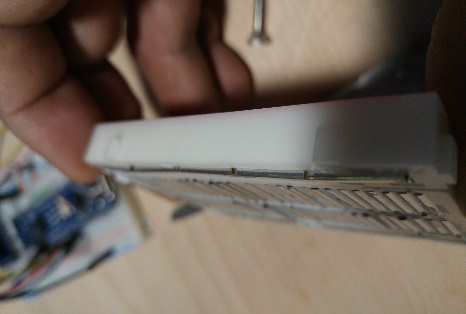


Figure Exposed connection strip showing thin "bridge" to be snipped

You just want enough of the strip out so you can see the thin metal strip that connects different segments of the power line.

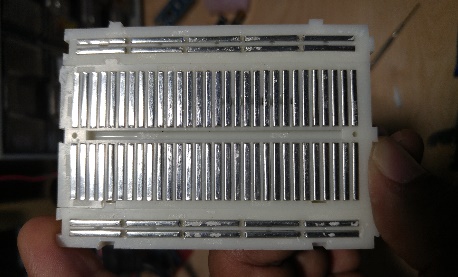


Figure After snipping bridges and pushing connectors back in place

I have snipped out the bridges and you can see how we now have separate sets of connections for each group of 5 tie-points.

Paste some adhesive felt sheets or double-side tape to cover the base of the breadboard and you are now in business with a breadboard with "increased capacity".

# Appendix B: Arduino Sketch

// esp8266\_test.ino

//

// Original:

// Plot LM35 data on thingspeak.com using an Arduino and an ESP8266 WiFi

// module.

//

// Author: Mahesh Venkitachalam

// Website: electronut.in

// adapted for my use

// -Venkat Swaminathan. Dec 2016

#include <SoftwareSerial.h>

#include <stdlib.h>

#include <OneWire.h>

#include <DallasTemperature.h>

// Data wire is plugged into pin 10 on the Arduino

#define ONE\_WIRE\_BUS 10

// Setup a oneWire instance to communicate with any OneWire devices

// (not just Maxim/Dallas temperature ICs)

OneWire oneWire(ONE\_WIRE\_BUS);

// Pass our oneWire reference to Dallas Temperature.

DallasTemperature sensors(&oneWire);

const int SENSORID = 1;

//const char \*tempToken = "temperature";

//const char \*lightToken = "light";

//const char \*distanceToken = "distance";

//const char \*motionToken = "motion";

// Define our pins

int ldrPin = A0; // analog LDR

int ldr2Pin = A1; // second light sensor

int serialRX = 2; // Serial port RX (to ESP8266); connect to TX on other side

int serialTX = 3; // Serial port TX (to ESP8266); connect to RX on other side

int motionPin = 4; // motion sensor data

int trigPin = 5; // trigger for HC-SR04 Ultrasonic sensor

int echoPin = 6; // received echo for HC-SR04 Ultrasonic Sensor

int buzzer = 8; // piezo buzzer

int motion2Pin = 9; // second motion sensor

int tempPin = 10; // data wire of DS18B20 onewire temperature sensor

int ledPin = 12; // heartbeat LED

int hiBriteLedPin = 13; // high intensity LED. Using default value due to how wiring is done

long timer1, timeInterval1; // timers to track stuff

long timer2, timeInterval2; // timers to track stuff

long lightSamplerTimer, temperatureSamplerTimer, motionSamplerTimer, distanceSamplerTimer;

long lightSampler2Timer, motionSampler2Timer;

long heartbeatTimer;

long distance = 0;

char jsonObj1[256];

float temperature,prevTemperature = 0.0, lightLevel, prevLightLevel = 0.0;

float lightLevel2, prevLightLevel2 = 0.0;

int motionLevel, prevMotionLevel = 0, prevMotionLevel2 = 0;

SoftwareSerial ser(serialRX, serialTX); // RX, TX

void blinkLED(int ledPin, boolean beep = false);

// this runs once

void setup() {

// initialize the digital pin as an output.

pinMode(ledPin, OUTPUT);

// enable debug serial

Serial.begin(9600);

// enable software serial

ser.begin(9600);

// Start up the library

sensors.begin();

timer1 = 10000; // 10 seconds for light?

timer2 = 30000; // milliseconds between measurements

lightSamplerTimer = millis();

lightSampler2Timer = millis();

temperatureSamplerTimer = millis();

motionSamplerTimer = millis();

motionSampler2Timer = millis();

distanceSamplerTimer = millis();

blinkLED(ledPin, true);

}

// the loop

void loop() {

//

// general principle of event emitter is to emit events

// on a timer, or if there is a large change in values

//

char temp[20] = ""; // use for conversions

// get temperature from DS18B20

sensors.requestTemperatures(); // Send the command to get temperatures

temperature = sensors.getTempCByIndex(0);

temperature = (temperature \* 9 / 5) + 32;

if ( millis() - temperatureSamplerTimer > timer2 || // timer triggered OR

abs(temperature - prevTemperature) > 1 ) { // more than a degree change in temperature

temp[0] = '\0';

dtostrf(temperature, 7, 2, temp);

jsonObj1[0] = '\0';

constructJSONObj("temperature", temp, jsonObj1);

sendDataOverSerial(jsonObj1);

temperatureSamplerTimer = millis();

prevTemperature = temperature;

}

// get light level from LDR

lightLevel = readLightLevel(ldrPin);

char temp1[20];

dtostrf(lightLevel, 9, 2, temp1);

if ( millis() - lightSamplerTimer > timer2 || // timer triggered, OR

abs( lightLevel - prevLightLevel) > 100) { // sudden change in light intensity

jsonObj1[0] = '\0';

constructJSONObj("light", temp1, jsonObj1);

sendDataOverSerial(jsonObj1);

lightSamplerTimer = millis();

prevLightLevel = lightLevel;

}

// get light level from LDR2

lightLevel = readLightLevel(ldr2Pin);

char temp2[20];

dtostrf(lightLevel, 9, 2, temp2);

if ( millis() - lightSampler2Timer > timer2 || // timer triggered, OR

abs( lightLevel - prevLightLevel2) > 100) { // sudden change in light intensity

jsonObj1[0] = '\0';

constructJSONObj("light2", temp2, jsonObj1);

sendDataOverSerial(jsonObj1);

lightSampler2Timer = millis();

prevLightLevel2 = lightLevel;

}

// get distance reading from PING sensor

// The sensor is triggered by a HIGH pulse of 10 or more microseconds.

// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

pinMode(trigPin, OUTPUT);

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

// Read the signal from the sensor: a HIGH pulse whose

// duration is the time (in microseconds) from the sending

// of the ping to the reception of its echo off of an object.

pinMode(echoPin, INPUT);

long duration = pulseIn(echoPin, HIGH);

// convert the time into a distance

distance = microsecondsToInches(duration);

// get proximity reading from PIR

motionLevel = digitalRead(motionPin);

if ( millis() - motionSamplerTimer > timer2 || // on timer, OR

motionLevel != prevMotionLevel ) { // when state changes

char temp2[3];

sprintf(temp2, "%d", motionLevel);

jsonObj1[0] = '\0';

constructJSONObj("motion", temp2, jsonObj1);

sendDataOverSerial(jsonObj1);

motionSamplerTimer = millis();

prevMotionLevel = motionLevel;

}

// get proximity reading from PIR2

motionLevel = digitalRead(motion2Pin);

if ( millis() - motionSamplerTimer > timer2 || // on timer, OR

motionLevel != prevMotionLevel2 ) { // when state changes

char temp2[3];

sprintf(temp2, "%d", motionLevel);

jsonObj1[0] = '\0';

constructJSONObj("motion2", temp2, jsonObj1);

sendDataOverSerial(jsonObj1);

motionSampler2Timer = millis();

prevMotionLevel2 = motionLevel;

}

// check for messages from ESP8266 that may have come in from MQTT

char message[255]; // 255 chars should be sufficient for everyone ;-)

boolean haveMessage = false;

int indx = 0;

if (ser.available()) {

haveMessage = true;

delay(1);

while ( ser.available() && indx < 255 ) {

char c = ser.read();

message[indx] = c;

indx++;

}

message[indx] = '\0'; // null terminate the string

}

if ( haveMessage) // if we do have a message then process it

processAction(message);

// heartbeat LED

if ( millis() - heartbeatTimer > 5000 ) {

blinkLED(ledPin);

heartbeatTimer = millis();

char infoMsg[50];

infoMsg[0] = '\0';

strcat(infoMsg, "INFO: SensorID ");

temp[0] = '\0';

sprintf(temp, "%d", SENSORID);

strcat(infoMsg, temp);

strcat(infoMsg, " is UP.");

strcat(infoMsg, "\0");

sendDataOverSerial(infoMsg);

}

}

float readLightLevel(int analogPin) {

// read the value from LDR.

// read 10 values for averaging.

int val = 0;

val += analogRead(analogPin);

return val;

}

void blinkLED(int pin, boolean beep) {

// blink LED on board

digitalWrite(pin, HIGH);

if ( beep ) {

tone( buzzer, 100 \* pin, 100);

}

delay(200);

digitalWrite(pin, LOW);

}

long microsecondsToInches(long microseconds)

{

// According to Parallax's datasheet for the PING))), there are

// 73.746 microseconds per inch (i.e. sound travels at 1130 feet per

// second). This gives the distance travelled by the ping, outbound

// and return, so we divide by 2 to get the distance of the obstacle.

// See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf

return microseconds / 74 / 2;

}

void sendDataOverSerial(char \*stringBuf) {

//Serial.println(stringBuf);

ser.print(stringBuf);

ser.flush();

delay(100);

}

void constructJSONObj( const char \*measurement, char \*value, char \*object) {

char temp[20] = ""; // use for conversions

object[0] = '\0'; // init string

strcat(object, "{\"SensorId\":\"");

sprintf(temp, "%d", SENSORID);

strcat(object, temp);

strcat(object, "\",\"Measurement\":\"");

strcat(object, measurement);

strcat(object, "\",\"Value\":\"");

strcat(object, value);

strcat(object, "\" }");

strcat(object, "\0");

}

//

// process actions that are sent to the ESP8266 either from MQTT

// or from the Arduino

//

// action strings are of the form: "ACTION,<sensorID>,<messageID>,<verb>,<object>,<value>" where

// <sensorID> = which sensor this message is meant for

// <messageID> = unique message ID possibly used for logging and ack

// <verb> = "SETVAR" or "SETPIN"

// <object> = one of "PIN","SENSORID","MQTTSRVR","SSID","PASSWORD"

// <value> = a string that corresponds to <object>, e.g. pin number or String.

//

void processAction(char \*actionString) {

char temp[20]; // use for conversions

char tempString[255];

strcpy(tempString, actionString);

char \*token;

char \*savePtr;

token = strtok\_r(tempString, ",", &savePtr);

//Serial.print("0<"); Serial.print(token); Serial.print(">");

if ( token != NULL && strcmp(token, "ACTION") == 0) { // found ACTION keyword

token = strtok\_r(NULL, ",", &savePtr);

//Serial.print("1<"); Serial.print(token); Serial.print(">");

int sensorID = atoi(token); // which sensor ID is this message meant for

if ( sensorID == SENSORID ) { // this is for me

token = strtok\_r(NULL, ",", &savePtr);

//Serial.print("2<"); Serial.print(token); Serial.print(">");

char messageID[10];

strcpy(messageID, token);

token = strtok\_r(NULL, ",", &savePtr); // get verb

//Serial.print("3<"); Serial.print(token); Serial.print(">");

if ( token != NULL && strcmp(token, "SETPIN") == 0) { // looks like we want to set a pin value

token = strtok\_r(NULL, ",", &savePtr);

//Serial.print("4<"); Serial.print(token); Serial.print(">");

if ( token != NULL) {

int pinID = atoi(token); // get which pin we want to modify

token = strtok\_r(NULL, ",", &savePtr);

//Serial.print("5<");Serial.print(token);Serial.print(">");

if ( token != NULL && strcmp(token, "HIGH") == 0) {

digitalWrite(pinID, HIGH);

// String outMessage = "INFO: Executed Action ID: " + messageID + ". ";

char ackMsg[80];

strcat(ackMsg, "INFO: Executed Action ID: ");

strcat(ackMsg, messageID);

strcat(ackMsg, "\0");

//Serial.println(outMessage);

sendDataOverSerial(ackMsg);

}

else if ( token != NULL && strcmp(token, "LOW") == 0) {

digitalWrite(pinID, LOW);

// String outMessage = "INFO: Executed Action ID: " + messageID + ". ";

char ackMsg[80] = "";

strcat(ackMsg, "INFO: Executed Action ID: ");

strcat(ackMsg, messageID);

strcat(ackMsg, "\0");

//Serial.println(outMessage);

sendDataOverSerial(ackMsg);

}

}

}

}

}

}

# Appendix C: ESP8266 Sketch

# /\*

# Basic ESP8266 MQTT example

# This sketch demonstrates the capabilities of the pubsub library in combination

# with the ESP8266 board/library.

# It connects to an MQTT server then:

# - publishes "hello world" to the topic "outTopic" every two seconds

# - subscribes to the topic "inTopic", printing out any messages

# it receives. NB - it assumes the received payloads are strings not binary

# - If the first character of the topic "inTopic" is an 1, switch ON the ESP Led,

# else switch it off

# It will reconnect to the server if the connection is lost using a blocking

# reconnect function. See the 'mqtt\_reconnect\_nonblocking' example for how to

# achieve the same result without blocking the main loop.

# To install the ESP8266 board, (using Arduino 1.6.4+):

# - Add the following 3rd party board manager under "File -> Preferences -> Additional Boards Manager URLs":

# http://arduino.esp8266.com/stable/package\_esp8266com\_index.json

# - Open the "Tools -> Board -> Board Manager" and click install for the ESP8266"

# - Select your ESP8266 in "Tools -> Board"

# \*/

# #include <ESP8266WiFi.h>

# #include <PubSubClient.h>

# // Update these with values suitable for your network.

# const char\* ssid = "ssid";

# const char\* password = "password";

# const char\* mqtt\_server = “ip address";

# char\* mgmtTopic = "SensorManagement";

# char\* sensorDataTopic = "SensorData";

# WiFiClient espClient;

# PubSubClient client(espClient);

# long lastMsg = 0;

# char msg[50];

# int value = 0;

# void setup() {

# pinMode(BUILTIN\_LED, OUTPUT); // Initialize the BUILTIN\_LED pin as an output

# Serial.begin(9600);

# setup\_wifi();

# client.setServer(mqtt\_server, 1883);

# client.setCallback(callback);

# }

# void setup\_wifi() {

# delay(10);

# // We start by connecting to a WiFi network

# Serial.println();

# Serial.print("Connecting to ");

# Serial.println(ssid);

# WiFi.mode(WIFI\_STA);

# WiFi.begin(ssid, password);

# while (WiFi.status() != WL\_CONNECTED) {

# delay(500);

# Serial.print(".");

# }

# Serial.println("");

# Serial.println("WiFi connected");

# Serial.println("IP address: ");

# Serial.println(WiFi.localIP());

# }

# void callback(char\* topic, byte\* payload, unsigned int length) {

# // Serial.print("Message arrived [");

# // Serial.print(topic);

# // Serial.print("] ");

# // for (int i = 0; i < length; i++) {

# // Serial.print((char)payload[i]);

# // }

# // Serial.println();

# //

# // // Switch on the LED if an 1 was received as first character

# // if ((char)payload[0] == '1') {

# // digitalWrite(BUILTIN\_LED, LOW); // Turn the LED on (Note that LOW is the voltage level

# // // but actually the LED is on; this is because

# // // it is acive low on the ESP-01)

# // } else {

# // digitalWrite(BUILTIN\_LED, HIGH); // Turn the LED off by making the voltage HIGH

# // }

# processAction((char\*) payload, length);

# }

# void reconnect() {

# // Loop until we're reconnected

# while (!client.connected()) {

# //Serial.print("Attempting MQTT connection...");

# // Attempt to connect

# if (client.connect("ESP8266Client88")) {

# //tempMessage = "Connected to network with IP " + WiFi.localIP().toString();

# char tempMessage[80] = "";

# strcat(tempMessage, "Connected to network with IP ");

# char myIPString2[24] = "";

# IPAddress myIP = WiFi.localIP();

# sprintf(myIPString2, "%d.%d.%d.%d", myIP[0], myIP[1], myIP[2], myIP[3]);

# strcat(tempMessage, myIPString2);

# client.publish(mgmtTopic, tempMessage);

# // ... and resubscribe

# client.subscribe("inTopic");

# } else {

# Serial.print("failed, rc=");

# Serial.print(client.state());

# Serial.println(" try again in 5 seconds");

# // Wait 5 seconds before retrying

# delay(5000);

# }

# }

# }

# void loop() {

# if (!client.connected()) {

# reconnect();

# }

# client.loop();

# char message[255] = "";

# int indx = 0;

# while (Serial.available() && indx < 255) {

# delay(1);

# if (Serial.available() > 0) {

# char c = Serial.read();

# message[indx] = c;

# indx++;

# }

# message[indx] = '\0';

# }

# if ( strstr(message, "INFO") != NULL) {

# char heartBeatMsg[335] = ""; // since message can be max of 255 and ack is max of 80

# strcat(heartBeatMsg, message);

# strcat(heartBeatMsg, " IP = ");

# char myIPString[24] = "";

# IPAddress myIP = WiFi.localIP();

# sprintf(myIPString, "%d.%d.%d.%d", myIP[0], myIP[1], myIP[2], myIP[3]);

# strcat(heartBeatMsg, myIPString);

# strcat(heartBeatMsg, "\0");

# client.publish(mgmtTopic, heartBeatMsg);

# }

# else {

# client.publish(sensorDataTopic, message);

# }

# }

# //

# // process actions that are sent to the ESP8266 either from MQTT

# // or (eventually) from the Arduino

# //

# void processAction(char \*actionString, int length) {

# char \*actionKeyword = "ACTION"; // forward to Arduino

# if ( strstr(actionString, actionKeyword) != NULL)

# for (int i = 0; i < length; i++) {

# Serial.print(actionString[i]); // sends to Arduino over serial port

# }

# Serial.flush();

# }