1. **DHT HUMIDITY AND TEMPERATURE:**

#include <dht.h>

#define dht\_apin A0

dht DHT;

void setup(){

Serial.begin(9600);

delay(500);

Serial.println("DHT11 Humidity & temperature Sensor\n\n");

delay(1000);

}

void loop(){

DHT.read11(dht\_apin);

Serial.print("Current humidity = ");

Serial.print(DHT.humidity);

Serial.print("% ");

Serial.print("temperature = ");

Serial.print(DHT.temperature);

Serial.println("C ");

delay(5000);

}

1. **BUZZER ALARM:**

// Declaring Pins

const int buzzerPin = 5;

const int ledPin = 6;

const int motionPin = 7;

const int buttonPin = 12;

// Setting Buzzer mode to False

boolean buzzer\_mode = false;

// For LED

int ledState = LOW;

long previousMillis = 0;

long interval = 100; // Interval at which LED blinks

void setup()

{

//The Following are our output

pinMode(ledPin,OUTPUT);

pinMode(buzzerPin,OUTPUT);

//Button is our Input

pinMode(buttonPin, INPUT);

// Wait before starting the alarm

delay(5000);

}

void loop()

{

// To chech whether the motion is detected or not

if (digitalRead(motionPin)) {

buzzer\_mode = true;

}

// If alarm mode is on,blink our LED

if (buzzer\_mode){

unsigned long currentMillis = millis();

if(currentMillis - previousMillis > interval) {

previousMillis = currentMillis;

if (ledState == LOW)

ledState = HIGH;

else

ledState = LOW;

// Switch the LED

digitalWrite(ledPin, ledState);

}

tone(buzzerPin,1000);

}

// If alarm is off

if (buzzer\_mode == false) {

// No tone & LED off

noTone(buzzerPin);

digitalWrite(ledPin, LOW);

}

// If our button is pressed Switch off ringing and Setup

int button\_state = digitalRead(buttonPin);

if (button\_state) {buzzer\_mode = false;}

}

1. **Connecting NodeMCU to WIFI:**

|  |
| --- |
|  |
|  | #include<ESP8266Wifi.h>  #include <PubSubClient.h> |
|  | #include <ArduinoJson.h> |
|  |  |
|  |  |
|  | const char\* ssid = "<yourWIFIssid>"; |
|  | const char\* password = "<yourWIFIpassword>"; |
|  |  |
|  | #define ORG "xyz1kg" |
|  | #define DEVICE\_TYPE "<Arduino" |
|  | #define DEVICE\_ID "<ard123" |
|  | #define TOKEN "12345678" |
|  |  |
|  |  |
|  | char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; |
|  | char authMethod[] = "use-token-auth"; |
|  | char token[] = TOKEN; |
|  | char clientId[] = "d:" ORG ":" DEVICE\_TYPE ":" DEVICE\_ID; |
|  |  |
|  | const char eventTopic[] = "iot-2/evt/status/fmt/json"; |
|  | const char cmdTopic[] = "iot-2/cmd/led/fmt/json"; |
|  |  |
|  |  |
|  |  |
|  | WiFiClient wifiClient; |
|  | void callback(char\* topic, byte\* payload, unsigned int payloadLength) { |
|  | Serial.print("Message arrived ["); |
|  | Serial.print(topic); |
|  | Serial.print("] "); |
|  | for (int i = 0; i < payloadLength; i++) { |
|  | Serial.print((char)payload[i]); |
|  | } |
|  | Serial.println(); |
|  |  |
|  | // Switch on the LED if an 1 was received as first character |
|  | if (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 |
|  | } |
|  |  |
|  | } |
|  | PubSubClient client(server, 1883, callback, wifiClient); |
|  |  |
|  | int publishInterval = 5000; // 5 seconds//Send adc every 5sc |
|  | long lastPublishMillis; |
|  |  |
|  | void setup() { |
|  | Serial.begin(9600); Serial.println(); |
|  | pinMode(LED\_BUILTIN, OUTPUT); |
|  | wifiConnect(); |
|  | mqttConnect(); |
|  | } |
|  |  |
|  | void loop() { |
|  | if (millis() - lastPublishMillis > publishInterval) { |
|  | publishData(); |
|  | lastPublishMillis = millis(); |
|  | } |
|  |  |
|  | if (!client.loop()) { |
|  | mqttConnect(); |
|  | } |
|  | } |
|  |  |
|  | void wifiConnect() { |
|  | Serial.print("Connecting to "); Serial.print(ssid); |
|  | WiFi.begin(ssid, password); |
|  | while (WiFi.status() != WL\_CONNECTED) { |
|  | delay(500); |
|  | Serial.print("."); |
|  | } |
|  | Serial.print("nWiFi connected, IP address: "); Serial.println(WiFi.localIP()); |
|  |  |
|  | } |
|  |  |
|  | void mqttConnect() { |
|  | if (!!!client.connected()) { |
|  | Serial.print("Reconnecting MQTT client to "); Serial.println(server); |
|  | while (!!!client.connect(clientId, authMethod, token)) { |
|  | Serial.print("."); |
|  | delay(500); |
|  | } |
|  | if (client.subscribe(cmdTopic)) { |
|  | Serial.println("subscribe to responses OK"); |
|  | } else { |
|  | Serial.println("subscribe to responses FAILED"); |
|  | } |
|  | Serial.println(); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | void publishData() { |
|  | // read the input on analog pin 0: |
|  | int sensorValue = analogRead(A0); |
|  |  |
|  | String payload = "{\"d\":{\"adc\":"; |
|  | payload += String(sensorValue, DEC); |
|  | payload += "}}"; |
|  |  |
|  | Serial.print("Sending payload: "); Serial.println(payload); |
|  |  |
|  | if (client.publish(eventTopic, (char\*) payload.c\_str())) { |
|  | Serial.println("Publish OK"); |
|  | } else { |
|  | Serial.println("Publish FAILED"); |
|  | } |
|  | } |

1. **For Publishing the data using MQTT:**

int Publish(char\* payload, int payload\_size) {  
  int rc = -1;  
  MQTTClient client = {0};  
  MQTTClient\_connectOptions conn\_opts = MQTTClient\_connectOptions\_initializer;  
  MQTTClient\_message pubmsg = MQTTClient\_message\_initializer;  
  MQTTClient\_deliveryToken token = {0};  
  
  MQTTClient\_create(&client, opts.address, opts.clientid,  
                    MQTTCLIENT\_PERSISTENCE\_NONE, NULL);  
  conn\_opts.keepAliveInterval = 60;  
  conn\_opts.cleansession = 1;  
  conn\_opts.username = k\_username;  
  conn\_opts.password = CreateJwt(opts.keypath, opts.projectid, opts.algorithm);  
  MQTTClient\_SSLOptions sslopts = MQTTClient\_SSLOptions\_initializer;  
  
  sslopts.trustStore = opts.rootpath;  
  sslopts.privateKey = opts.keypath;  
  conn\_opts.ssl = &sslopts;  
  
  unsigned long retry\_interval\_ms = kInitialConnectIntervalMillis;  
  unsigned long total\_retry\_time\_ms = 0;  
  while ((rc = MQTTClient\_connect(client, &conn\_opts)) != MQTTCLIENT\_SUCCESS) {  
    if (rc == 3) {  // connection refused: server unavailable  
      usleep(retry\_interval\_ms \* 1000);  
      total\_retry\_time\_ms += retry\_interval\_ms;  
      if (total\_retry\_time\_ms >= kMaxConnectRetryTimeElapsedMillis) {  
        printf("Failed to connect, maximum retry time exceeded.");  
        exit(EXIT\_FAILURE);  
      }  
      retry\_interval\_ms \*= kIntervalMultiplier;  
      if (retry\_interval\_ms > kMaxConnectIntervalMillis) {  
        retry\_interval\_ms = kMaxConnectIntervalMillis;  
      }  
    } else {  
      printf("Failed to connect, return code %d\n", rc);  
      exit(EXIT\_FAILURE);  
    }  
  }  
  
  pubmsg.payload = payload;  
  pubmsg.payloadlen = payload\_size;  
  pubmsg.qos = kQos;  
  pubmsg.retained = 0;  
  MQTTClient\_publishMessage(client, opts.topic, &pubmsg, &token);  
  printf(  
      "Waiting for up to %lu seconds for publication of %s\n"  
      "on topic %s for client with ClientID: %s\n",  
      (kTimeout / 1000), opts.payload, opts.topic, opts.clientid);  
  rc = MQTTClient\_waitForCompletion(client, token, kTimeout);  
  printf("Message with delivery token %d delivered\n", token);  
  MQTTClient\_disconnect(client, 10000);  
  MQTTClient\_destroy(&client);  
  
  return rc;  
}