Tcp client

```
#include"ESP8266WiFi.h"
#include"DHT.h"
#define DHTTYPE DHT11
const char* ssid = "ASP";
const char* password =  "123456789";
WiFiServer wifiServer(9000);
DHT dht(D4, DHT11);
void setup(){
  Serial.begin(115200);
  delay(1000);
  WiFi.begin(ssid, password);

  while(WiFi.status() != WL_CONNECTED){
    delay(1000);
    Serial.println("Connecting..");
  }
  Serial.print("Connected to WiFi. IP:");
  Serial.println(WiFi.localIP());
  wifiServer.begin();
  dht.begin();
}

void loop(){
   WiFiClient client = wifiServer.available();
```

```
  if(client){
    while(client.connected()){
      while(client.available()>0){
        float t=dht.readTemperature();
        float h = dht.readHumidity();
        client.print("humidity :");
        client.print("temperature :");
        client.println(h);
        Serial.println(h);
        client.println(t);
        Serial.println(t);
        delay(2000);
      }
    }
    client.stop();
    Serial.println("Client disconnected");
  }
}
```

UDP SERVER

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <DHT.h>
const char* ssid ="Anu";
const char* password = "123456789";
const char* udpAddress = "192.168.0.7";
const int udpPort = 8081;
#define DHTPIN D4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
WiFiUDP udp;
void setup() {
 Serial.begin(115200);
 Serial.println();
 Serial.println("Connecting to WiFi...");
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(2000);
  Serial.print(".");
 }
 Serial.println();
 Serial.println("WiFi connected.");
 dht.begin();
```

```
}
void loop() {
  delay(2000);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C\tHumidity: ");
  Serial.print(humidity);
  Serial.println(" %");
  Serial.println("Sending data over UDP...");
  udp.beginPacket(udpAddress, udpPort);
  udp.print("Temperature: ");
  udp.print(temperature);
  udp.print(" °C, Humidity: ");
  udp.print(humidity);
  udp.println(" %");
  udp.endPacket();
  Serial.println("Data sent over UDP.");
}
```

8.DHT 11 DATA RETRIEVE

```
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
const char ssid[] = "Anu";  // your network SSID (name)
const char pass[] = "123456789";   // your network password
int statusCode = 0;
WiFiClient  client;

//---------Channel Details---------//
unsigned long counterChannelNumber = 2845211;           // Channel ID
const char * myCounterReadAPIKey = "DF7LO78W7FHP3668"; // Read API Key
const int FieldNumber1 = 1;  // The field you wish to read
const int FieldNumber2 = 2;  // The field you wish to read
//-------------------------------//

void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);
}

void loop()
{
  //----------------- Network -----------------//
```

```
 if (WiFi.status() != WL_CONNECTED)
 {
  Serial.print("Connecting to ");
  Serial.print(ssid);
  Serial.println(" ....");
  while (WiFi.status() != WL_CONNECTED)
  {
    WiFi.begin(ssid, pass);
    delay(5000);
  }
  Serial.println("Connected to Wi-Fi Succesfully.");
 }
 //--------- End of Network connection--------//


 long temp = ThingSpeak.readLongField(counterChannelNumber, FieldNumber1,
myCounterReadAPIKey);
 statusCode = ThingSpeak.getLastReadStatus();
 if (statusCode == 200)
 {
  Serial.print("Temperature: ");
  Serial.println(temp);
 }
 else
 {
  Serial.println("Unable to read channel / No internet connection");
```

```
  }
  delay(100);



  long humidity = ThingSpeak.readLongField(counterChannelNumber, FieldNumber2,
myCounterReadAPIKey);
  statusCode = ThingSpeak.getLastReadStatus();
  if (statusCode == 200)
  {
    Serial.print("Humidity: ");
    Serial.println(humidity);
  }
  else
  {
    Serial.println("Unable to read channel / No internet connection");
  }
  delay(100);


}
```

7. DHT 11 DATA TO CLOUD

```
#include <ESP8266WiFi.h>

#include "secrets.h"

#include "ThingSpeak.h"

#include "DHT.h"


#define DHTPIN D4

#define DHTTYPE DHT11// always include thingspeak header file after other header files and custom macros


char ssid[] = SECRET_SSID;   // your network SSID (name)

char pass[] = SECRET_PASS;   // your network password

int keyIndex = 0;            // your network key Index number (needed only for WEP)

WiFiClient  client;


unsigned long myChannelNumber = SECRET_CH_ID;

const char * myWriteAPIKey = SECRET_WRITE_APIKEY;


// Initialize our values

int number1 = 0;

int number2 = random(0,100);

int number3 = random(0,100);

int number4 = random(0,100);

String myStatus = "";


void setup() {
```

```
 Serial.begin(115200);  // Initialize serial
 while (!Serial) {
   ; // wait for serial port to connect. Needed for Leonardo native USB port only
 }

 WiFi.mode(WIFI_STA);
 ThingSpeak.begin(client);  // Initialize ThingSpeak
}

void loop() {

 // Connect or reconnect to WiFi
 if(WiFi.status() != WL_CONNECTED){
   Serial.print("Attempting to connect to SSID: ");
   Serial.println(SECRET_SSID);
   while(WiFi.status() != WL_CONNECTED){
     WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network. Change this line if using open or WEP network
     Serial.print(".");
     delay(5000);
   }
   Serial.println("\nConnected.");
 }

 // set the fields with the values
 ThingSpeak.setField(1, number1);
```

```
ThingSpeak.setField(2, number2);
ThingSpeak.setField(3, number3);
ThingSpeak.setField(4, number4);

// figure out the status message
if(number1 > number2){
  myStatus = String("field1 is greater than field2");
}
else if(number1 < number2){
  myStatus = String("field1 is less than field2");
}
else{
  myStatus = String("field1 equals field2");
}

// set the status
ThingSpeak.setStatus(myStatus);

// write to the ThingSpeak channel
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if(x == 200){
  Serial.println("Channel update successful.");
}
else{
  Serial.println("Problem updating channel. HTTP error code " + String(x));
```

```
  }


  // change the values
  number1++;
  if(number1 > 99){
    number1 = 0;
  }
  number2 = random(0,100);
  number3 = random(0,100);
  number4 = random(0,100);


  delay(20000); // Wait 20 seconds to update the channel again
}
```

6.IR SENSOR TO CLOUD

```
#include <ESP8266WiFi.h>
#include "secrets.h"
#include "ThingSpeak.h" // always include thingspeak header file after other header files and custom macros


char ssid[] = SECRET_SSID;   // your network SSID (name)
char pass[] = SECRET_PASS;   // your network password
int keyIndex = 0;            // your network key Index number (needed only for WEP)
WiFiClient  client;
```

```
unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;

int number = 0;
const int irpin =D4;

void setup() {
  Serial.begin(115200);  // Initialize serial
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo native USB port only
  }

  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
}

void loop() {

  // Connect or reconnect to WiFi
  if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(SECRET_SSID);
    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network. Change this line if using open or WEP
network
```

```
    Serial.print(".");
    delay(5000);
  }
  Serial.println("\nConnected.");
 }

 // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8 different
 // pieces of information in a channel.  Here, we write to field 1.
 int x = ThingSpeak.writeField(myChannelNumber, 1, number, myWriteAPIKey);
 if(x == 200){
  Serial.println("Channel update successful.");
 }
 else{
  Serial.println("Problem updating channel. HTTP error code " + String(x));
 }

 // change the value
 number++;
 if(number > 99){
  number = 0;
 }

 delay(20000); // Wait 20 seconds to update the channel again
}
```

5.DHT TESTER

```
#include "DHT.h"


#define DHTPIN 2     // Digital pin connected to the DHT sensor

// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --

// Pin 15 can work but DHT must be disconnected during program upload.


// Uncomment whatever type you're using!

#define DHTTYPE DHT11   // DHT 11

//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

//#define DHTTYPE DHT21   // DHT 21 (AM2301)


// Connect pin 1 (on the left) of the sensor to +5V

// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1

// to 3.3V instead of 5V!

// Connect pin 2 of the sensor to whatever your DHTPIN is

// Connect pin 3 (on the right) of the sensor to GROUND (if your sensor has 3 pins)

// Connect pin 4 (on the right) of the sensor to GROUND and leave the pin 3 EMPTY (if your sensor has 4 pins)

// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor


// Initialize DHT sensor.

// Note that older versions of this library took an optional third parameter to

// tweak the timings for faster processors.  This parameter is no longer needed

// as the current DHT reading algorithm adjusts itself to work on faster procs.

DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
```

```
 // Compute heat index in Fahrenheit (the default)
 float hif = dht.computeHeatIndex(f, h);
 // Compute heat index in Celsius (isFahreheit = false)
 float hic = dht.computeHeatIndex(t, h, false);

 Serial.print(F("Humidity: "));
 Serial.print(h);
 Serial.print(F("%  Temperature: "));
 Serial.print(t);
 Serial.print(F("°C "));
 Serial.print(f);
 Serial.print(F("°F  Heat index: "));
 Serial.print(hic);
 Serial.print(F("°C "));
 Serial.print(hif);
 Serial.println(F("°F"));
}
```

4.BLUETOOTH

```
#include <SoftwareSerial.h>
SoftwareSerial Bluetooth(9, 8); // RX, TX
int LED = 4; // the on-board LED
int Data; // the data received
```

```
void setup() {
  Bluetooth.begin(9600);
  Serial.begin(9600);
  Serial.println("Waiting for command...");
  Bluetooth.println("Send 1 to turn on the LED. Send 0 to turn Off");
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Bluetooth.available()){ //wait for data received
Data=Bluetooth.read();
 if(Data=='1'){
digitalWrite(LED,1);
Serial.println("LED On!");
Bluetooth.println("LED On!");
    }
    else if(Data=='0'){
```

```
digitalWrite(LED,0);
 Serial.println("LED Off!");
Bluetooth.println("LED  On D7 Off ! ");
    }
  }
delay(100);
}
```

MFRC-522

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN         A5          // Configurable, see typical pin layout above
#define SS_PIN          10     // Configurable, see typical pin layout above
MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance
void setup() {
Serial.begin(115200);                  // Initialize serial communications with the PC
while (!Serial);                  // Do nothing if no serial port is opened (added for Arduinos based on ATMEGA32U4)
        SPI.begin();                      // Init SPI bus
        mfrc522.PCD_Init();           // Init MFRC522
        delay(4);                                 // Optional delay. Some board do need more time after init to be ready, see Readme
        mfrc522.PCD_DumpVersionToSerial();     // Show details of PCD - MFRC522 Card Reader details
```

```
	Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}

void loop() {
	// Reset the loop if no new card present on the sensor/reader. This saves the entire process when idle.
	if ( ! mfrc522.PICC_IsNewCardPresent()) {
		return;
	}
	// Select one of the cards
	if ( ! mfrc522.PICC_ReadCardSerial()) {
		return;
	}
	// Dump debug info about the card; PICC_HaltA() is automatically called
	mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```