EnviroSense Touch

1. Yaghini

## Table of contents

[Table of contents](#_y4w4rs7dojhv)

[Components](#_zcna5gigiriw)

[Wemose 2.4 inch Touch](#_a68x2xm5e7o9)

[SGP30](#_c77aver9tun)

[WEMOS SHT30](#_chrocsivgozt)

[Code:](#_uti8jv61zpdn)

# Components

## Wemose 2.4 inch Touch

Product page:

<https://www.wemos.cc/en/latest/d1_mini_shield/tft_2_4.html>

The example page has some useful functions that can be used to develop the code:

<https://github.com/wemos/D1_mini_Examples/tree/master/examples/04.Shields/TFT_2.4_Touch_Shield>

Test code from github

[https://github.com/wemos/D1\_mini\_Examples/tree/master/examples/04.Shields/TFT\_2.4\_Touch\_Shield]:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. !!!Make sure you using latest ESP8266 core for Arduino, otherwise it may not work properly.

https://github.com/esp8266/Arduino

(The pin D0(GPIO16) will be not work when you use older version ESP8266 core for Arduino,

because the older version ESP8266 core for Arduino's digitalPinToBitMask(), portOutputRegister(),

portInputRegister() and portModeRegister() function have some bugs which Adafruit\_ILI9341 Library will use.

This bug was fixed after commit: https://github.com/esp8266/Arduino/commit/799193888a553de8876052019842538396f92194 )

2. Setup latest Adafruit\_GFX, Adafruit\_ILI9341 and XPT2046\_Touchscreen Library first:

https://github.com/adafruit/Adafruit-GFX-Library

https://github.com/adafruit/Adafruit\_ILI9341

https://github.com/PaulStoffregen/XPT2046\_Touchscreen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <SPI.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_ILI9341.h>

#include <XPT2046\_Touchscreen.h>

#define TFT\_CS D0 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TFT\_DC D8 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TFT\_RST -1 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TS\_CS D3 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

// #define TFT\_CS 14 //for D32 Pro

// #define TFT\_DC 27 //for D32 Pro

// #define TFT\_RST 33 //for D32 Pro

// #define TS\_CS 12 //for D32 Pro

Adafruit\_ILI9341 tft = Adafruit\_ILI9341(TFT\_CS, TFT\_DC, TFT\_RST);

XPT2046\_Touchscreen ts(TS\_CS);

void setup()

{

Serial.begin(115200);

ts.begin();

ts.setRotation(1);

tft.begin();

tft.setRotation(3);

tft.fillScreen(ILI9341\_BLACK);

tft.setTextColor(ILI9341\_WHITE);

tft.setTextSize(3);

tft.println("Touch Test");

while (!Serial && (millis() <= 1000))

;

}

void loop()

{

if (ts.touched())

{

TS\_Point p = ts.getPoint();

tft.fillScreen(ILI9341\_BLACK);

tft.setCursor(0, 0);

tft.print("Pressure = ");

tft.println(p.z);

tft.print("X = ");

tft.println(p.x);

tft.print("Y = ");

tft.println(p.y);

}

}

## SGP30

Product page: <https://www.wemos.cc/en/latest/d1_mini_shield/sgp30.html>

## WEMOS SHT30

Product page: <https://www.wemos.cc/en/latest/d1_mini_shield/sht30.html>

# Code:

//-----INCLUDES LIBRARIES---------------------------------------------------------

//--GENERAL LIBS

#include <Credentials.h>

#include <Wire.h>

#include <SPI.h>

#include <NTPClient.h>

#include <ESP8266WiFi.h>

#include <WiFiUdp.h>

#include <ArduinoJson.h>

#include <WiFiClient.h>

#include <ESP8266mDNS.h>

#include <ArduinoOTA.h>

#include "PubSubClient.h" // Connect and publish to the MQTT broker

//--WEMOS TFT 2.4 Touch----> https://www.wemos.cc/en/latest/d1\_mini\_shield/tft\_2\_4.html

#include <Adafruit\_GFX.h>

#include <Adafruit\_ILI9341.h>

#include <XPT2046\_Touchscreen.h>

//--WEMOS SGP30 CO2---->https://www.wemos.cc/en/latest/d1\_mini\_shield/sgp30.html

#include "Adafruit\_SGP30.h"

//--WEMOS SHT30

#include <WEMOS\_SHT3X.h> //needs to be downloaded from: https://github.com/wemos/WEMOS\_SHT3x\_Arduino\_Library

//--------------------------------------------------------------------------------

//-----DEFENITIONS-----

//--TFT 2.4

#define TFT\_CS D0 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TFT\_DC D8 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TFT\_RST -1 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

#define TS\_CS D3 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

Adafruit\_ILI9341 tft = Adafruit\_ILI9341(TFT\_CS, TFT\_DC, TFT\_RST);

XPT2046\_Touchscreen ts(TS\_CS);

//--SGP30

Adafruit\_SGP30 sgp;

bool isSgp;

int sgpCounter;

float tVoc = 0, eCo2 = 0, lasttVoc, lasteCo2;

//--SHT30

SHT3X sht30(0x45);

float temp = 0, humid = 0, lastTemp, lastHumid;

//+++++++++SETUP PARAMETERS++++++++++

const char\* ssid = mySSID;

const char\* password = myPASSWORD;

int state = 0;

//---------> TIME INETRVALS <---------------

long updateTimeInterval = 30000; // interval at which to send (milliseconds)

long previousMillis = updateTimeInterval; // will store last time data was sent

float displayThereshold = 20.0;

//---------> NTP Client <---------------

const long utcOffsetInSeconds = -25200;

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

String timeToDisplay = "waiting to receive the time from server";

//---------> Weather Map API <---------

WiFiClient client;

const char server[] = "api.openweathermap.org";

String nameOfCity = "Vancouver,CA";

String apiKey = "b7abf0ceb4e2d3b297c96282778e94de";

String weatherText;

int jsonend = 0;

boolean startJson = false;

int status = WL\_IDLE\_STATUS;

#define JSON\_BUFF\_DIMENSION 2500

unsigned long lastConnectionTime = 30 \* 60 \* 1000; // last time you connected to the server, in milliseconds

const unsigned long postInterval = 30 \* 60 \* 1000; // posting interval of 10 minutes (10L \* 1000L; 10 seconds delay for testing)

bool weatherUpdated = false;

String weatherForcast = "Updating..";

//---------> MQTT <---------

const char\* mqtt\_server = "192.168.1.92"; // IP of the MQTT broker

const char\* humidity\_topic = "home/EnviroSenseTouch/humidity";

const char\* temperature\_topic = "home/EnviroSenseTouch/temperature";

const char\* tVoc\_topic = "home/EnviroSenseTouch/tVoc";

const char\* eCO2\_topic = "home/EnviroSenseTouch/equvalentCO2";

//const char\* mqtt\_username = "cdavid"; // MQTT username

//const char\* mqtt\_password = "cdavid"; // MQTT password

const char\* clientID = "client\_EnviroSenseTouch"; // MQTT client ID

PubSubClient mqttClient(mqtt\_server, 1883, client);

long intervalMQTT = 5 \* 60 \* 1000, lastTimeSentMQTT = intervalMQTT;

//--------------------------------------------------------------------------------

void setup() {

InitializeDisplay();

InitializeWifi();

isSgp = InitializeSgp();

ArduinoOTAInitializer();

delay(1000);

StartApp();

delay(1000);

}

void loop() {

ArduinoOTA.handle();

if (ts.touched()) HandleTouch();

StateMachine();

}

//--------------------------------------------------------------------------------

void StateMachine() {

unsigned long currentMillis = millis();

switch (state) {

case 0: {

if (isSgp) MeasureSgp();

state++;

break;

}

case 1: {

if (currentMillis - previousMillis > updateTimeInterval) {

previousMillis = currentMillis;

UpdateTime();

DisplayManager();

}

state++;

break;

}

case 2: {

MeasureSht();

state++;

break;

}

case 3: {

if (currentMillis - lastConnectionTime > postInterval) {

lastConnectionTime = currentMillis;

UpdateWeather();

}

state ++;

break;

}

case 4: {

if (currentMillis - lastTimeSentMQTT > intervalMQTT) {

lastTimeSentMQTT = currentMillis;

SendMQTT();

}

state++;

break;

}

case 5: {

UpdateDisplay();

state = 0;

break;

}

}

}

void HandleTouch() {

TS\_Point p = ts.getPoint();

tft.fillScreen(ILI9341\_BLACK);

tft.setCursor(0, 0);

tft.print("Pressure = ");

tft.println(p.z);

tft.print("X = ");

tft.println(p.x);

tft.print("Y = ");

tft.println(p.y);

}

void UpdateTime() {

timeClient.update();

timeToDisplay = timeClient.getFormattedTime();

// UpdateDisplay();

}

void MeasureSgp() {

if (! sgp.IAQmeasure()) {

// Serial.println("Measurement failed");

return;

}

lasttVoc = tVoc;

lasteCo2 = eCo2;

tVoc = sgp.TVOC;

eCo2 = sgp.eCO2;

// if (! sgp.IAQmeasure()) {

//// Serial.println("Measurement failed");

//// return;

// }

// tft.print("TVOC "); tft.print(sgp.TVOC); tft.print(" ppb\t");

// tft.print("eCO2 "); tft.print(sgp.eCO2); tft.println(" ppm");

//

// if (! sgp.IAQmeasureRaw()) {

//// Serial.println("Raw Measurement failed");

//// return;

// }

// tft.print("Raw H2 "); tft.print(sgp.rawH2); tft.print(" \t");

// tft.print("Raw Ethanol "); tft.print(sgp.rawEthanol); tft.println("");

//

// delay(1000);

//

// sgpCounter++;

// if (sgpCounter == 30) {

// sgpCounter = 0;

//

// uint16\_t TVOC\_base, eCO2\_base;

// if (! sgp.getIAQBaseline(&eCO2\_base, &TVOC\_base)) {

// tft.println("Failed to get baseline readings");

// return;

// }

// tft.print("\*\*\*\*Baseline values: eCO2: 0x"); tft.print(eCO2\_base, HEX);

// tft.print(" & TVOC: 0x"); tft.println(TVOC\_base, HEX);

// }

}

void MeasureSht() {

if (sht30.get() == 0) {

temp, humid, lastTemp, lastHumid;

lastTemp = temp;

lastHumid = humid;

temp = sht30.cTemp;

humid = sht30.humidity;

// Serial.print("Temperature in Celsius : ");

// Serial.println(sht30.cTemp);

// Serial.print("Temperature in Fahrenheit : ");

// Serial.println(sht30.fTemp);

// Serial.print("Relative Humidity : ");

// Serial.println(sht30.humidity);

// Serial.println();

}

else

{

// Serial.println("Error!");

}

}

void UpdateWeather() {

weatherUpdated = false;

weatherForcast = WehtherRequest();

}

void UpdateDisplay() {

float t = Difrentiator(temp, lastTemp);

float h = Difrentiator(humid, lastHumid);

float v = Difrentiator(tVoc, lasttVoc);

float c = Difrentiator(eCo2, lasteCo2);

if (t >= displayThereshold || h >= displayThereshold || c > +displayThereshold) {

DisplayManager();

// tft.fillScreen(ILI9341\_BLACK);

// tft.setCursor(0, 0);

// tft.setTextColor(ILI9341\_GREEN);

// tft.print("Temp : ");

// tft.print(temp);

// tft.println("c");

// tft.print("Humid: ");

// tft.print(humid);

// tft.println("%");

// tft.setTextColor(ILI9341\_YELLOW);

// tft.print("tVOC : ");

// tft.print(tVoc);

// tft.println("ppb");

// tft.print("eCO2 : ");

// tft.print(eCo2);

// tft.println("ppm");

// tft.println();

// tft.println(timeToDisplay);

// tft.println();

// tft.println(weatherForcast);

}

}

void DisplayManager() {

int cx = tft.width() / 3 - 1,

cy = tft.height() / 4 - 1;

tft.fillScreen(ILI9341\_BLACK);

tft.setCursor(0, 0);

DrawButton(cx / 3, cy / 2, cx, cy, String(temp, 1), ColorMeter(temp,20,25,27));

DrawButton(5 \* cx / 3, cy / 2, cx, cy, String(humid, 1), ColorMeter(humid,40,70,80));

DrawButton(cx / 3, 5 \* cy / 2, cx, cy, String(eCo2, 0), ColorMeter(eCo2,400,800,1000));

DrawButton(5 \* cx / 3, 5 \* cy / 2, cx, cy, String(tVoc, 0), ColorMeter(tVoc,50,500,1000));

tft.setTextColor(ILI9341\_WHITE);

PrintCentreString(timeToDisplay, tft.width() / 2, cy / 4);

tft.setCursor(0, 0);

PrintCentreString(weatherForcast, tft.width() / 2, tft.height() -cy/10);

//PrintCentreString("CLOUDY", tft.width() / 2, tft.height() -cy/10);

}

uint16\_t ColorMeter(int n, int low, int high, int extreme) {

if (n < low) {

return ILI9341\_BLUE;

} else if (n < high) {

return ILI9341\_GREEN;

} else if (n < extreme) {

return ILI9341\_ORANGE;

} else {

return ILI9341\_RED;

}

}

void DrawButton(int x, int y, int cx, int cy, String text, uint16\_t color) {

int r = 4;

int t = 3;

uint16\_t bc = ILI9341\_BLACK;

int textSize = 3;

tft.fillRoundRect(x, y, cx, cy, r, color);

tft.fillRoundRect(x + t, y + t, cx - t - t, cy - t - t, r, bc);

tft.setTextColor(color);

tft.setTextSize(textSize);

PrintCentreString(text, x + cx / 2, y + cy / 2);

}

void PrintCentreString(const String &t, int x, int y)

{

int16\_t x1, y1;

uint16\_t w, h;

tft.getTextBounds(t, x, y, &x1, &y1, &w, &h); //calc width of new string

tft.setCursor(x - w / 2, y - h / 2);

tft.print(t);

}

void SendMQTT() {

// Connect to MQTT Broker

// client.connect returns a boolean value to let us know if the connection was successful.

// If the connection is failing, make sure you are using the correct MQTT Username and Password (Setup Earlier in the Instructable)

if (mqttClient.connect(clientID)) {

SendMQTTMessages(temperature\_topic, String(temp));

SendMQTTMessages(humidity\_topic, String(humid));

SendMQTTMessages(tVoc\_topic, String(tVoc));

SendMQTTMessages(eCO2\_topic, String(eCo2));

mqttClient.disconnect(); // disconnect from the MQTT broker

}

else {

//Serial.println("Connection to MQTT Broker failed...");

}

}

void SendMQTTMessages(const char\* topic, String parameter) {

if (!mqttClient.publish(topic, parameter.c\_str())) {

mqttClient.connect(clientID);

delay(10); // This delay ensures that client.publish doesn't clash with the client.connect call

mqttClient.publish(topic, parameter.c\_str());

}

}

float Difrentiator(float a, float b) {

float r = ((a - b) / a) \* 100;

return r;

}

void InitializeDisplay() {

ts.begin();

ts.setRotation(1);

tft.begin();

tft.setRotation(3);

tft.fillScreen(ILI9341\_BLACK);

tft.setTextColor(ILI9341\_WHITE);

tft.setTextSize(3);

tft.println("EnviroSense Touch");

tft.println("A. Yaghini");

}

void InitializeWifi() {

tft.println("");

Progressor("WiFi");

WiFi.begin(ssid, password);

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

delay ( 500 );

//Serial.print ( "." );

}

tft.println(" ok");

}

bool InitializeSgp() {

Progressor("SGP30");

if (! sgp.begin()) {

tft.setTextColor(ILI9341\_RED);

tft.println(" X");

tft.setTextColor(ILI9341\_WHITE);

return false;

} else {

tft.setTextColor(ILI9341\_GREEN);

tft.println(" ok");

tft.setTextColor(ILI9341\_WHITE);

return true;

}

}

void Progressor(String in) {

tft.print(in);

delay(200);

tft.setTextColor(ILI9341\_RED);

tft.print(".");

delay(200);

tft.print("..");

delay(200);

tft.print("...");

tft.setTextColor(ILI9341\_WHITE);

}

void StartApp() {

Progressor("Time");

timeClient.begin();

tft.setTextColor(ILI9341\_GREEN);

tft.println(" ok");

tft.setTextColor(ILI9341\_WHITE);

delay(1000);

Progressor("Weather");

weatherText.reserve(JSON\_BUFF\_DIMENSION);

tft.setTextColor(ILI9341\_GREEN);

tft.println(" ok");

tft.setTextColor(ILI9341\_WHITE);

delay(3000);

tft.fillScreen(ILI9341\_GREEN);

}

void ArduinoOTAInitializer() {

ArduinoOTA.setHostname("Enviro\_Touch");

ArduinoOTA.begin();

}

// to request data from OWM

String WehtherRequest() {

// close any connection before send a new request to allow client make connection to server

client.stop();

// if there's a successful connection:

if (client.connect(server, 80)) {

// Serial.println("connecting...");

// send the HTTP PUT request:

client.println("GET /data/2.5/forecast?q=" + nameOfCity + "&APPID=" + apiKey + "&mode=json&units=metric&cnt=2 HTTP/1.1");

client.println("Host: api.openweathermap.org");

client.println("User-Agent: ArduinoWiFi/1.1");

client.println("Connection: close");

client.println();

unsigned long timeout = millis();

while (client.available() == 0) {

if (millis() - timeout > 5000) {

Serial.println(">>> Client Timeout !");

client.stop();

return "CONNECTION!";

}

}

char c = 0;

while (client.available()) {

c = client.read();

// since json contains equal number of open and close curly brackets, this means we can determine when a json is completely received by counting

// the open and close occurences,

//Serial.print(c);

if (c == '{') {

startJson = true; // set startJson true to indicate json message has started

jsonend++;

}

if (c == '}') {

jsonend--;

}

if (startJson == true) {

weatherText += c;

}

// if jsonend = 0 then we have had received equal number of curly braces

if (jsonend == 0 && startJson == true) {

//parseJson(weatherText.c\_str()); // parse c string text in parseJson function

StaticJsonDocument<JSON\_BUFF\_DIMENSION> doc;

DeserializationError error = deserializeJson(doc, weatherText);

if (error) {

//Serial.print(F("deserializeJson() failed: "));

//Serial.println(error.c\_str());

return "JSON ERR";

}

//int id = doc["weather"][0]["id"];

int id = doc["list"][0]["weather"][0]["id"];

float temp = doc["list"][0]["main"]["feels\_like"];

//more infor regarding id codes: https://openweathermap.org/weather-conditions

//int id = doc["cod"];

//Serial.println(weatherText);

//Serial.println(id);

String condition = "UPDATING";

if (id > 100) {

if (id >= 200 & id < 300) {

condition = "THUNDERSTORM";

}

if (id >= 300 & id < 400) {

condition = "DRIZZLE";

}

if (id >= 500 & id < 600) {

condition = "RAIN";

}

if (id >= 600 & id < 700) {

condition = "SNOW";

}

if (id >= 700 & id < 800) {

condition = "ATMOSPHERE";

}

if (id == 800) {

condition = "CLEAR";

}

if (id > 800) {

condition = "CLOUDY";

}

//Serial.println(condition);

weatherUpdated = true;

}

else {

weatherUpdated = false;

break;

}

weatherText = ""; // clear text string for the next time

startJson = false; // set startJson to false to indicate that a new message has not yet started

String sep = " ";

String unit = "c";

// String report = condition + sep + temp + unit;

String report = condition + " " + String(temp,1) + "c";

return report;

}

}

}

else {

// if no connction was made:

//Serial.println("connection failed");

//return "NAS";

}

}