Explication de l’émetteur version 7

## Présentation générale

Cette version du code émetteur est conçue pour fonctionner avec l'appareil M5Stack, utilisant le module ENVII comprenant un capteur de pression BMP280 et un capteur d'humidité et de température SHT30, le tout sur une carte Arduino ESP32. Le programme s'initialise au démarrage du M5Stack, configure les capteurs, se connecte au réseau Wi-Fi, puis entre dans une boucle principale. Dans cette boucle, il lit la température et calcule le Humidex, envoie la température lus ainsi que le Humidex calculé, génère des messages d'alerte en cas de dépassement des seuils définis, chiffre les messages qu’il envoie et met à jour sont affichage ainsi que les données des capteurs toutes les 5 secondes.

*Note : le capteur de pression BMP280 n’est pas utilisé à proprement parler, il est uniquement utilisé pour alimenter le capteur de température et humidité SHT30 qui se trouve avec lui dans le module ENVII.*

*Remarque : nous utilisons un kit M5Stack comprenant plusieurs capteurs, le kit M5Go.*

Sommaire

[**Présentation générale 1**](#_cvzux1v6fepa)

[**Fonction setup() 4**](#_otg4lwblyr4t)

[M5.begin(true, false, true, true); 4](#_kdqm1cozuhpn)

[initSensors(); 4](#_z6kioue9jyj8)

[Fonction initSensors() 4](#_ihvajvcqtxlf)

[bmp.begin(0x76); 4](#_kid224fsu5n3)

[//sht.init(); 4](#_6d7zkmgextor)

[M5.Lcd.setTextColor(TFT\_WHITE, TFT\_BLACK); 5](#_4rhvfk53xwoz)

[M5.Lcd.setTextDatum(MC\_DATUM); 5](#_n8vrg8e3gqb3)

[Serial.begin(115200); 5](#_ukepzb2gten5)

[Fonction connectToWiFi() 5](#_bz06hahoyh6f)

[La boucle while… 6](#_a4ezamf2bgb)

[M5.Lcd.println("\nConnecte au reseau WiFi"); 6](#_wlwxu9i7spaa)

[**Fonction loop() 6**](#_m1usnihikzbu)

[float temperature = readTemperature(); 7](#_aqrsw0unossu)

[Fonction readTemperature() 7](#_upfks490mc52)

[sht.get(); 7](#_asfe7zka8azg)

[return sht.cTemp; 7](#_e8znogryibct)

[float humindex = calculateHumindex(); 8](#_2mteasng8m39)

[Fonction calculateHumindex() 8](#_fjvdu4g615xu)

[float T = sht.cTemp; 8](#_o16ii1mekgk5)

[float HR = sht.humidity; 8](#_aznyvu7un678)

[float es = 6.112 \* exp((17.67 \* T) / (T + 243.5)); 8](#_eudedgesn7im)

[float e = (HR / 100.0) \* es; 8](#_u4nk5c22y3pm)

[float humidex = T + (5.0 / 9.0) \* (e - 10.0); 8](#_yy820kmqkzhf)

[return humidex; 9](#_5hrhf8zdhh7g)

[Le if-else… 9](#_v4v72xbo4xfj)

[encryptedMessageToSend = messageEncryption(messageToSend); 9](#_f8oreoch35ie)

[Fonction messageEncryption() 9](#_hns7c0nd9lwl)

[String cryptedMessage = ""; 10](#_p56drurrsw6w)

[La boucle for (char currentChar : messageToEncrypt) { ... } 10](#_2q4cxiemjkyj)

[if (currentChar == '\0') { cryptedMessage += '\0'; } 10](#_68pe1a1bho2t)

[cryptedMessage += (char)(asc + 26 \* ((asc < 65) - (asc > 90))); 10](#_u1257p17qche)

[cryptedMessage += endSignatureMarker; 10](#_50sz0ndg9bmd)

[return cryptedMessage; 10](#_g8sfu1vhj693)

[WiFiClient client; 11](#_blbe79g2nrku)

[Le if… 11](#_ak75cbjiq61w)

[delay(5000); 11](#_k25s21zadl32)

[**Annexe 12**](#_qmhbomlyt1cj)

[Capture d’écran du rapport de compilation du code de l’émetteur/capteur M5Stack version 7 par l’IDE Arduino version 2.2.1. 12](#_plaift7512ij)

## 

## Fonction setup()

void setup() {

M5.begin(true, false, true, true); // Initialisation de M5Stack avec écran activé, son désactivé, vibreur activé et charge de la batterie activée

initSensors(); // Initialisation des capteurs

connectToWiFi(); // Connexion au réseau Wi-Fi

}

#### M5.begin(true, false, true, true);

Cette ligne initialise la bibliothèque M5Stack. Les paramètres indiquent si l'écran, le son, le vibreur et la charge de la batterie doivent être activés ou désactivés.

#### **initSensors();**

Appelle la fonction **initSensors()** qui initialise les capteurs.

### Fonction **initSensors()**

void initSensors() {

bmp.begin(0x76); // Initialisation du capteur de pression avec l'adresse I2C

//sht.init(); // Initialisation du capteur d'humidité et de température

M5.Lcd.setTextColor(TFT\_WHITE, TFT\_BLACK); // Configuration des couleurs de l'écran

M5.Lcd.setTextDatum(MC\_DATUM); // Configuration du point de référence du texte

Serial.begin(115200); // Initialisation de la communication série

}

#### bmp.begin(0x76);

Initialise le capteur de pression (bmp) avec l'adresse I2C spécifiée (0x76).

#### //sht.init();

Cette ligne permet d’initialiser le capteur de température et humidité, je l’ai commenté parce que “bmp.begin(0x76);” permet déjà d’initialiser l’ensemble du module ENVII dans lequel il y a à la fois le capteur de pression BMP280 et le capteur de température et humidité SHT30.

#### M5.Lcd.setTextColor(TFT\_WHITE, TFT\_BLACK);

Configure les couleurs de l'écran LCD de M5Stack avec du texte blanc sur fond noir.

#### M5.Lcd.setTextDatum(MC\_DATUM);

Configure le point de référence du texte sur le centre de l'écran.

#### Serial.begin(115200);

Initialise la communication série/communication pour le débuggage à une vitesse de 115200 bauds.

En résumé, la fonction **initSensors** initialise le capteur de pression, configure les paramètres de l'écran et démarre la communication série.

1. **connectToWiFi();** Appelle la fonction **connectToWiFi()** qui gère la connexion au réseau Wi-Fi.En résumé, la fonction setup initialise l'appareil M5Stack, initialise les capteurs et se connecte au réseau Wi-Fi au démarrage du programme.

### Fonction **connectToWiFi()**

void connectToWiFi() {

WiFi.begin(ssid, password); // Connexion au réseau Wi-Fi

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

delay(1000);

M5.Lcd.print(".");

}

M5.Lcd.println("\nConnecte au reseau WiFi");

}

WiFi.begin(ssid, password);

Initie la connexion au réseau Wi-Fi en utilisant le nom du réseau (ssid) et le mot de passe (password).

#### La boucle while…

…attend que la connexion soit établie. Pendant cette attente, elle affiche des points sur l'écran LCD de M5Stack.

#### M5.Lcd.println("\nConnecte au reseau WiFi");

Affiche sur l'écran LCD que la connexion au réseau Wi-Fi a été établie.

En résumé, la fonction **connectToWiFi** gère la connexion au réseau Wi-Fi et informe l'utilisateur lorsque la connexion est établie.

## Fonction loop()

void loop() {

float temperature = readTemperature(); // Lecture de la température

float humindex = calculateHumindex(); // On calcul et récupère le Humindex

if (temperature > maxTemperature) {

messageToSend = createAlertMessageTemperature(temperature); // Création d'un message d'alerte

} else if (humindex > maxHumindex) {

messageToSend = createAlertMessageHumindex(humindex); // Création d'un message d'alerte

} else {

messageToSend = createRegularMessage(temperature, humindex); // Création d'un message régulier

}

encryptedMessageToSend = messageEncryption(messageToSend);

WiFiClient client; // Création d'un client Wi-Fi

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

sendAndDisplayMessage(client); // Envoi du message au récepteur et affichage sur l'émetteur

} else {

handleConnectionError(); // Gestion de l'erreur de connexion

}

delay(5000); // Attente de 5 secondes avant la prochaine itération/envoi de message

}

#### float temperature = **readTemperature();**

Lit la température à l'aide de la fonction **readTemperature()** et stocke le résultat dans la variable temperature.

### Fonction readTemperature()

float readTemperature() {

sht.get(); // Obtenir les données du capteur d'humidité et de température

return sht.cTemp;

}

#### sht.get();

Appelle la fonction get() du capteur d'humidité et de température (sht) pour obtenir les données actuelles.

(La fonction get() est une fonction de base de la classe SHT. Elle permet d'obtenir les données actuelles du capteur d'humidité et de température.

Dans le cas de l'instruction sht.get();, la fonction get() est appelée sur l'objet sht. L'objet sht représente un capteur d'humidité et de température.

La fonction get() retourne un objet SHTData qui contient les données actuelles du capteur. Ces données sont les suivantes :

* *temperature*: la température en degrés Celsius
* *humidity*: l'humidité relative en pourcentage
* *status*: le statut du capteur

L'instruction sht.get(); peut être utilisée pour lire les données du capteur à tout moment.)

#### return sht.cTemp;

Renvoie la température en degrés Celsius (cTemp) du capteur d'humidité et de température.

En résumé, **readTemperature()** récupère la température à partir du capteur d'humidité et de température et la renvoie.

#### float humindex = **calculateHumindex();**

Calcule et récupère le Humindex à l'aide de la fonction **calculateHumindex()** et le stocke dans la variable humindex.

### Fonction calculateHumindex()

float calculateHumindex() {

float T = sht.cTemp;

float HR = sht.humidity;

float es = 6.112 \* exp((17.67 \* T) / (T + 243.5));

float e = (HR / 100.0) \* es;

float humidex = T + (5.0 / 9.0) \* (e - 10.0);

return humidex;

}

#### float T = sht.cTemp;

Stocke la température en degrés Celsius (cTemp) du capteur dans la variable T.

#### float HR = sht.humidity;

Stocke l'humidité relative (humidity) du capteur dans la variable HR.

#### float es = 6.112 \* exp((17.67 \* T) / (T + 243.5));

Calcule la pression de vapeur saturante (es) à partir de la température.

#### float e = (HR / 100.0) \* es;

Calcule la pression de vapeur d'eau effective (e) en fonction de l'humidité relative.

#### float humidex = T + (5.0 / 9.0) \* (e - 10.0);

Calcule le Humidex à partir de la température et de la pression de vapeur d'eau effective.

#### return humidex;

Renvoie la valeur calculée du Humidex.

En résumé, **calculateHumindex()** utilise les données du capteur d'humidité et de température pour calculer et renvoyer le Humidex, un indice combinant la température et l'humidité relative.

#### Le if-else…

…vérifie si la température dépasse la température maximale autorisée (maxTemperature).

1. Si c'est le cas, il crée un message d'alerte (**createAlertMessageTemperature**).
2. Sinon, il vérifie si le Humindex dépasse le Humindex maximal autorisé (maxHumindex) et crée un message d'alerte correspondant. Si aucune condition n'est satisfaite, il crée un message régulier.

#### encryptedMessageToSend = **messageEncryption(**messageToSend**)**;

Chiffre le message à l'aide de la fonction **messageEncryption()** et stocke le résultat dans encryptedMessageToSend.

### Fonction messageEncryption()

String messageEncryption(String messageToEncrypt) {

// Chaîne pour stocker le message chiffré

String cryptedMessage = "";

// Boucle à travers chaque caractère du message à chiffrer

for (char currentChar : messageToEncrypt) {

// Si le caractère actuel est nul, ajoute un caractère nul au message chiffré

if (currentChar == '\0') {

cryptedMessage += '\0';

} else {

// Sinon, effectue l'opération de chiffrement

int asc = (int)currentChar + cryptingKey;

cryptedMessage += (char)(asc + 26 \* ((asc < 65) - (asc > 90)));

}

}

// Ajoute le marqueur de fin de signature à la fin du message chiffré

cryptedMessage += endSignatureMarker; // À mettre en variable globale

// Renvoie le message chiffré

return cryptedMessage;

}

**String messageEncryption(**String messageToEncrypt**)**

La fonction **messageEncryption** prend une chaîne de caractères **messageToEncrypt** en entrée et renvoie la chaîne chiffrée résultante.

#### String cryptedMessage = "";

Initialise une chaîne vide cryptedMessage qui stockera le message chiffré.

#### La boucle for (char currentChar : messageToEncrypt) { ... }

Traverse chaque caractère de la chaîne messageToEncrypt.

#### if (currentChar == '\0') { cryptedMessage += '\0'; }

Si le caractère actuel est nul, ajoute un caractère nul au message chiffré.

Sinon, effectue l'opération de chiffrement :int asc = (int)currentChar + cryptingKey;

#### cryptedMessage += (char)(asc + 26 \* ((asc < 65) - (asc > 90)));

Convertit le caractère actuel en valeur ASCII ((int)currentChar) et ajoute la clé de chiffrement (cryptingKey). Soit utilise une opération de chiffrement simple qui décale les caractères alphabétiques vers le haut de l'alphabet. Assure que les lettres restent dans la plage ASCII alphabétique. Nous utilisons le chiffrement de César.

#### cryptedMessage += endSignatureMarker;

Ajoute le marqueur de fin de signature à la fin du message chiffré.

#### return cryptedMessage;

Renvoie le message chiffré.

Cette fonction prend chaque caractère du message, le chiffre selon une méthode de substitution simple, et ajoute un marqueur de fin de signature qui permettra au récepteur de savoir où la signature est pour authentifier l’émetteur, mais aussi pour savoir où est le message à afficher. Elle est utilisée pour crypter les messages avant de les envoyer au M5Stack récepteur/Dispositif porté.

Grâce à cette fonction la signature d’authentification, la température et l’Humidex sont chiffrées d’une manière simple, efficace et légère.le code doit être léger en terme de calcul et de poids puisque nous somme limité en autonomie de batterie et espace de stockage cf. en annexe stockage utilisé par la version 7.

#### WiFiClient client;

Créer un client Wi-Fi.

#### Le if…

…tente de se connecter au serveur avec l'adresse IP spécifiée (serverIP) sur le port 80.

1. Si la connexion réussit, il appelle la fonction **sendAndDisplayMessage(client)** pour envoyer le message au récepteur et afficher le message sur l'émetteur.
2. Sinon, il appelle la fonction **handleConnectionError()** pour gérer les erreurs de connexion.

#### delay(5000);

Ajoute une pause de 5 secondes avant la prochaine itération/envoi de message. On peut dire que c’est notre temps de rafraîchissement des données du capteur.

# Annexe



##### *Capture d’écran du rapport de compilation du code de l’émetteur/capteur M5Stack version 7 par l’IDE Arduino version 2.2.1.*

**Le texte en entier :**

Le croquis utilise 854893 octets (65%) de l'espace de stockage de programmes. Le maximum est de 1310720 octets.

Les variables globales utilisent 45648 octets (13%) de mémoire dynamique, ce qui laisse 282032 octets pour les variables locales. Le maximum est de 327680 octets.