Groupe 1 : Nathan LEROY, Emile PANCHOUT, Erwan MARTIN, Antoine PILLON.

Projet Worldwide Weather Watcher

Livrable 4: Documentation

Objectif : Présenter les opérés, informer l'utilisateur des performances du système.

I. Présentation générale

Le projet Worldwide Weather Watcher est une station météo embarqué faite pour être transporté par des navires de surveillances afin de capter et mesurer les paramètres influant sur la formation de cyclone ou tout autre catastrophe naturelle.

La station est composée d'une carte Arduino, de deux modules adaptables (respectivement le module Shield permettant un accès ergonomique aux ports de la carte, et un module pour la carte SD afin de permettre l'archivage des mesures) ainsi que les 5 capteurs de l'environnement (l'horloge, le capteur de luminosité, de température, d'hygrométrie et de pression atmosphérique) et enfin, deux boutons permettant le passage entre les modes (seul l'un des deux bouton servira car l'utilisation des deux boutons en même temps crée des interférence).

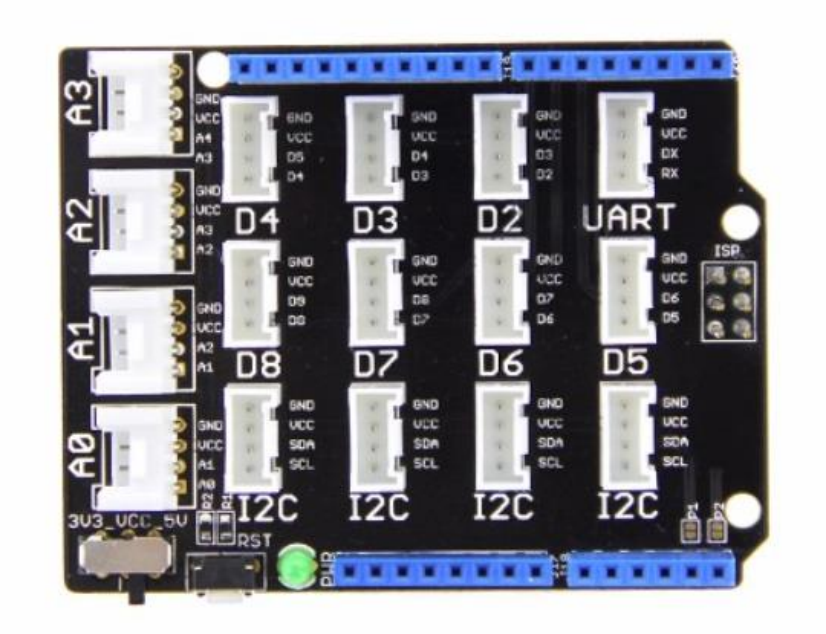
II. Documentation technique :

1. Composants :

- Carte Arduino :

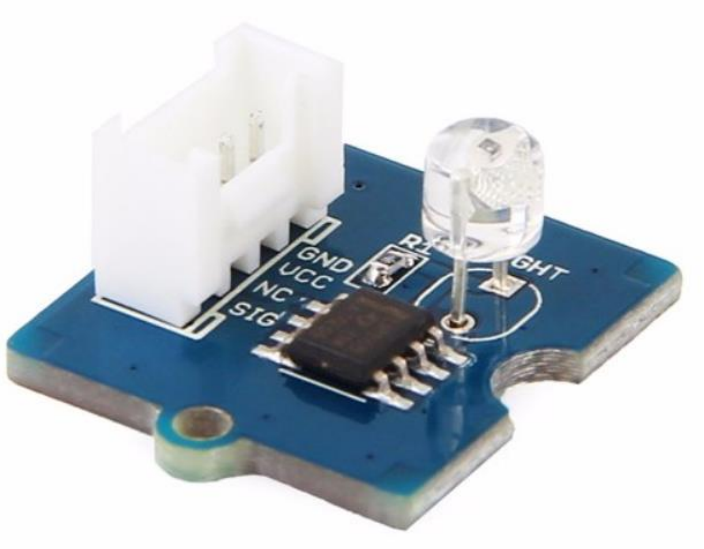
Le microcontrôleur Arduino UNO contient le programme à exécuter et met en lien tous les périphériques.

- Arduino Base Shield :



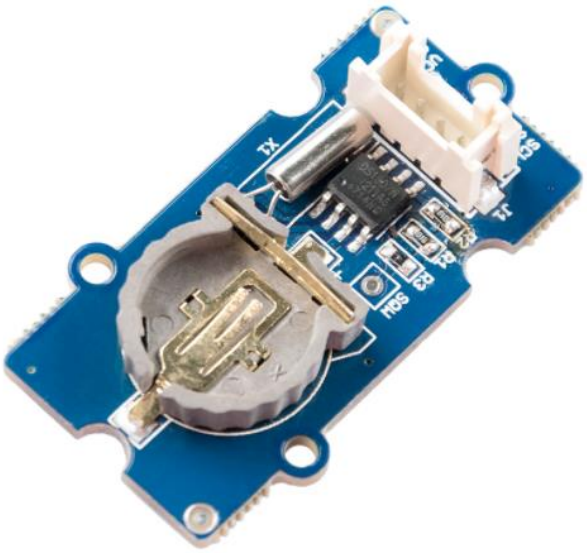
L'Arduino Shield sert à simplifier le montage électrique.

- Les capteurs :

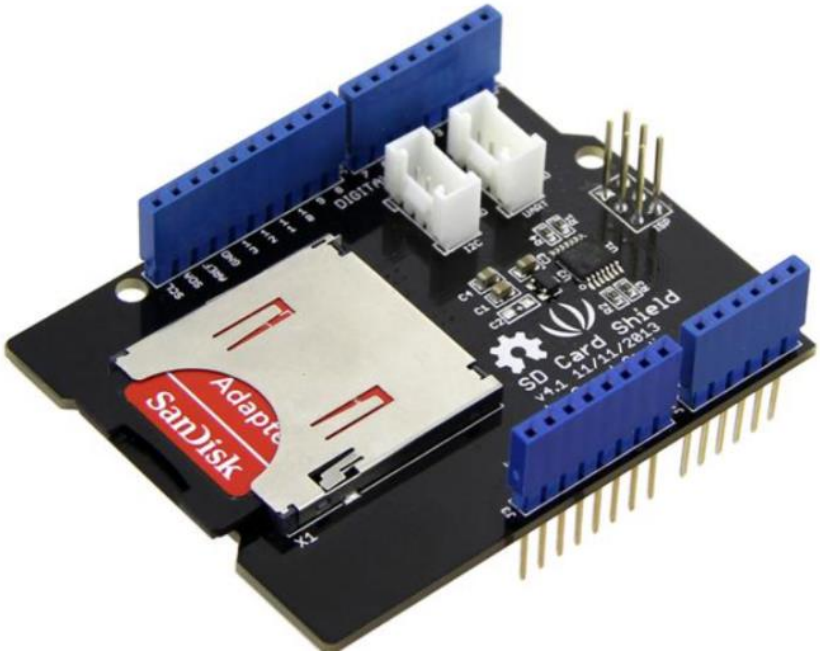


Le capteur de Luminosité (va renvoyer la mesure de luminosité en signal analogique, se branche sur un des ports Analogiques du Shield.)

Le capteur de température, humidité et pression. Se branche sur un des ports I2C du Shield.

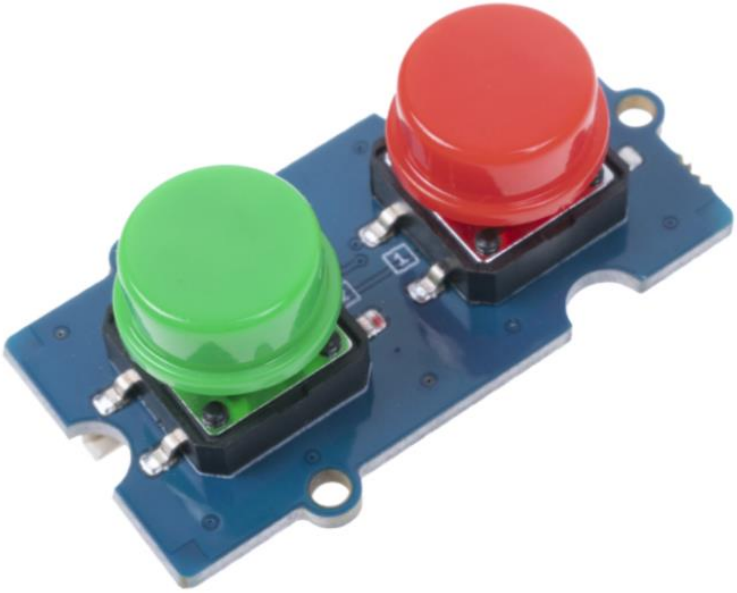


L’horloge RTC permet d’obtenir l’heure une fois initialisée, on la branche sur un port I2C du Shield.



Le SD Card Shield permet de lier notre carte microSD à l’Arduino.

La ChainableLED RGB s'allume en fonction de la couleur paramétrée, ainsi on verra dans quel mode nous sommes. On branche la LED à un port digital du Shield.



Le Dual Button permet de déclencher des interruptions, on le branche au port Digital2 du Shield (D2).

2. Description du Code

- prise de mesures :

(le code est joint avec ce fichier)

```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <BME280I2C.h>
#include "DS1307.h"
#include "recoltedonnees.h"
#include "saveSD.h"
#define SERIAL_BAUD 9600

BME280I2C bme;
DS1307 clock;

int FILE_SIZE_MAX = 4096;



void setup(){
    Serial.begin(SERIAL_BAUD);
    Wire.begin();
    clock.begin();
    clock.fillByYMD(2020,10,21);
    clock.fillByHMS(15,59,10);
    //clock.fillDayOfWeek(WED);
    clock.setTime();
    initSD();
    while(!bme.begin())
    {
        Serial.println("Could not find BME280 sensor!");
        delay(1000);
    }


}
void loop(){
    Mesures prise;
    prise = prisemesures(bme, clock);
    saveSD(&prise);
    delay(6000);
}
```

Voici le main(ci-dessus), dans le setup on initialise l'horloge et le capteur BME.

Dans le loop, on prend et on place dans une structure nos mesures grâce a l'appel de la fonction

Afin de stocker nos mesures, on crée une structure permettant de tout stocker en une variable (on la stock dans la bibliotheque).

```
typedef struct Mesures Mesures;
     struct Mesures{
          int heure;
          int minute;
          int sec;
          int jour;
          int mois;
          int annee;
          float lumin;
          float pression;
          float temperature;
          float humidite;
     };
```

Notre Fonction ***prisemesures*** qu'on appelle du *main* va retourner une structure de type « Mesures »:

```
#include <Wire.h>
#include <BME280I2C.h>
#include "DS1307.h"
#include "recoltedonnees.h"
#define SERIAL_BAUD 9600



void priseheure(Mesures *prise, DS1307 clock){
    clock.getTime();
    prise->heure = clock.hour;
    prise->minute = clock.minute;
    prise->sec = clock.second;
    prise->mois = clock.month;
    prise->annee = clock.year;
}

Mesures prisemesures(BME280I2C bme, DS1307 clock){
    float temp(NAN), hum(NAN), pres(NAN);
    Mesures prise;// on créé notre structure prise

    BME280::TempUnit tempUnit(BME280::TempUnit_Celsius);
    BME280::PresUnit presUnit(BME280::PresUnit_Pa);

    bme.read(pres,temp,hum,tempUnit,presUnit);
    prise.pression = pres;
    prise.temperature = temp;
    prise.humidite = hum;
    prise.lumin = analogRead(A0);
    priseheure(&prise, clock);

    return (prise);
}
```

On lit les mesures des capteurs de luminosité, pression, température et humidité.

La fonction ***priseheure*** va remplir l'heure dans notre structure(en mettant l'adresse de la variable complexe en paramètre d'appel, on va mettre en place un pointeur permettant de modifier directement les valeurs de notre variable).

Pour finir on renvoi la structure créée localement afin de l'enregistrer dans une variable du même type dans le *main*.

La fonction ***saveSD***, qu'on appelle du main après avoir pris et stocké nos mesures :

```
#include <Wire.h>
#include <SD.h>
#include <BME280I2C.h>
#include "DS1307.h"
#include <recoltedonnees.h>
#include <saveSD.h>


void initSD(){
    const byte SDCARD_CS_PIN = 4;
    pinMode(10, OUTPUT);

    Serial.print(F("Init SD card..."));
    if(!SD.begin(SDCARD_CS_PIN)){
        Serial.println(F("Fail appuyez sur reset"));
        for(;;);//appui sur reset
    }
    Serial.println(F("OK"));
}

void saveSD(Mesures *prise){
    int nbfile = 0;
    File myfile;
    char filename[14];
    File root = SD.open("/");
    File testfile = root.openNextFile();

 myfile = SD.open("/fichier1.txt", FILE_WRITE);
     myfile.print("temperature:");
     myfile.print(prise->temperature);
     myfile.print("/");
     myfile.print("pression:");
     myfile.print(prise->pression);
     myfile.print("/");
     myfile.print("humidite:");
     myfile.print(prise->humidite);
     myfile.print("/");
     myfile.print("luminosité:");
     myfile.print(prise->lumin);
     myfile.print("/");
     myfile.print("heure:");
     myfile.print(prise->heure);
     myfile.print(":");
     myfile.print(prise->minute);
     myfile.print(":");
     myfile.print(prise->sec);
     myfile.print("/");
     myfile.print("Date:");
     myfile.print(prise->jour);
     myfile.print("/");
     myfile.print(prise->mois);
     myfile.print("/");
     myfile.println(prise->annee);
     myfile.close();
     Serial.println("mesures rentrées");
```

On crée un fichier dans la carte SD à l'aide de la bibliothèque « SD.h » s'il n'existe pas.

Puis, on écrit nos données dans ce fichier créé.

Afin de permettre le changement de mode ainsi que l'affichage LED, on utilise les interruptions attachées aux boutons(le code est joint avec ce fichier).

On a pour cela inclue la bibliothèque associée aux LED et définie les ports d'entrée et de sorties du système. Le bouton est branché sur le pin digital 2, et la LED sur le pin 8.

On initialise en amont l'interruption et cette interruption fera appelle à la fonction Changer pour permettre de mesurer la durée de la pression du le bouton avec la variable « sec » grâce à la fonction ***millis***().

```
pinMode(button1,INPUT);
attachInterrupt(digitalPinToInterrupt(button1), Changer, CHANGE);

 void Changer(){
 int secb = 0;
 if(sec1==0){
 sec1 = millis();
 }else{
 sec2 = millis();
 secb=sec2-sec1;
 sec1=0;
 if(secb<100){
 sec1=millis();
 }else{
 sec=secb;
```

En plus de cela, on définit la variable « mode » qui indique au programme dans quel mode il se trouve.

Grâce à cela, on peut alors colorer la LED de la bonne couleur.

```
// On change la couleur de la led en fonction du mode actuel
if(mode == 0){leds.setColorRGB(0, 0, 0, 0);}
if(mode == 1){leds.setColorRGB(0, 0, 255, 0);}
if(mode == 2){leds.setColorRGB(0, 255, 255, 0);}
if(mode == 3){leds.setColorRGB(0, 0, 0, 255);}
if(mode == 4){leds.setColorRGB(0, 255, 70, 0);}
```

Pour ce qui est des changements de mode, on vérifie dans quel mode se trouvait le système avant la pression du bouton et combien de temps on appuie.

Pour rappel nous n'utilisons qu'un bouton sur les deux à cause d'interférence avec le deuxième bouton. Une pression comprise entre 1 et 5 secondes équivaut à appuyer dur le bouton rouge, une pression de plus de 5s équivaut à appuyer sur le bouton vert.

```
// on change le mode en fonction des pressions du bouton
if(mode == 0){
if(sec >= 1000 && sec <= 5000 && mode == 0){mode = 1; sec = 0;}
if(sec >= 5000 && mode == 0){mode = 2; sec = 0;}
}

else if(mode == 1){
if(sec >= 1000 && sec <= 5000 && mode == 1){mode = 4; sec = 0;mode_pre = 1;}
if(sec >= 5000 && mode == 1){mode = 3; sec = 0;mode_pre = 1;}
}
else if(mode == 3){
if(sec >= 1000 && sec <= 5000 && mode == 3){mode = 4; sec =0;mode_pre = 3;}
if(sec >= 5000 && mode == 3){mode = 1; sec = 0;mode_pre = 3;}
}
else if(mode == 4){
if(sec >= 1000 && sec <= 5000 && mode == 4 && mode_pre == 1){mode = 1; sec = 0; mode_pre = 4;}
if(sec >= 1000 && sec <= 5000 && mode == 4 && mode_pre == 3){mode = 3; sec = 0; mode_pre = 4;}
}
```

Pour le mode configuration :
Voici le code sur Thinkercad :
https://www.tinkercad.com/things/es4dKBp1QQE-mode-configuration/editel?sharecode=HI93GManR1AF36N4NdnyqF0CxHcmsmhfl7PlBt4dBO0

Le loop débute avec une variable « timer » qui prend la valeur de millis(), une fonction qui renvoie le nombre de millisecondes depuis lequel le programme s'est lancé. Si le timer atteint 1 800 000 secondes, c'est-à-dire 30 minutes, le programme bascule en mode standard.

Le second bloc détecte quand l'Arduino reçoit des données afin de scanner la commande entrée par l'utilisateur. Pour cela, il lit chaque caractère entrant et stocke sa valeur ASCII (exemple : F = 70) dans un tableau. Ensuite, la boucle for permet d'afficher chaque valeur du tableau (si on enlève cette boucle, le programme rencontre des problèmes).

```
void loop() {
  bool affichage = false;
  int incomingData = 0;
  int i;
  int n_max = 13;

  timer = millis();

  if (timer == 1800000) {
    //basculer en mode standard
  }

  // envoyer des données seulement quand Arduino en reçoit
  if (Serial.available() > 0) {
    // lire l'octet entrant
    incomingData = Serial.read();
    ascii[n] = incomingData;
    n++;

    for (i = 0; i < n_max; i++) {
      Serial.println(ascii[i]);
    }
  }
  else {
    affichage = true;
  }
```

Ensuite, une série de « if » permet de détecter la commande entrée en vérifiant les valeurs de tableau ASCII (exemple : ascii[0] == 82 && ascii[4] == 84 détecte la commande RESET car c'est la seule commande qui a en premier caractère « R » et en cinquième caractère « T »). Le booléen affichage permet d'empêcher des erreurs d'affichage.

```
if (ascii[0] == 70 && ascii[12] == 69 && affichage) {
  Serial.println("FILE_MAX_SIZE");
  affichage = false;
  FILE_MAX_SIZE();
  clear_tab();
}
else if (ascii[0] == 82 && ascii[4] == 84 && affichage) {
  Serial.println("RESET");
  affichage = false;
  RESET();
  clear_tab();
}
else if (ascii[0] == 86 && ascii[12] == 0 && affichage) {
  Serial.println("VERSION");
  affichage = false;
  VERSION();
  clear_tab();
}
else if (ascii[0] == 84 && ascii[1] == 73 && ascii[12] ==
```

Quand la commande est détectée, elle s'affiche sur l'interface, affichage passe en false, la fonction associée à la commande s'exécute :

- Les variables et les fonctions de toutes les commandes sont déclarées. Chaque fonction permet de demander à l'utilisateur la valeur qu'il souhaite attribuer à la commande, grâce à la fonction userMessage() et l'affiche ensuite.

```
void FILE_MAX_SIZE() {
  String file_max_size = "";
  bool test = false;

  if (test == false) {
    Serial.println("Valeur de FILE_MAX_SIZE (Valeur par defaut: 4096 ; Plage: {0;4096}): ");

    while (file_max_size == "") {
      file_max_size = userMessage();
      test = true;
    }
  }

  Serial.print("FILE_MAX_SIZE=");
  Serial.println(file_max_size);
}
```

- La fonction userMessage() attend que l'Arduino reçoive des données pour les lire et les stocker dans la variable « charac » puis dans « reponse » qui est ensuite retournée.

```
String userMessage() {
  String reponse = "";
  char charac = '0';

  if (Serial.available() > 0) {
    while (Serial.available() > 0) {
      charac = Serial.read();
      reponse += charac;
      delay(2);
    }

    return reponse;
  }

  return "";
}
```

Et la fonction clear_tab() permet de réinitialiser les valeurs du tableau afin de pouvoir entrer une autre commande ensuite : pour chaque case du tableau, on définit sa valeur à 0.

```
int clear_tab() {
  for (n=0; n < 13; n++) {
    ascii[n] = 0;
  }
  n = 0;
}
```

<u>Points à améliorer :</u>

-Dans le code des mesures :

- Il y avait pour la sauvegarde SD un début de code pour sauvegarder sur plusieurs fichiers, que nous avons mis en commentaire afin de simplifier sur une sauvegarde sur un seul fichier.
- L'horloge a des soucis au niveau de la prise de données pour le jour.

-Dans le code de la configuration :

- les commandes CLOCK, DATE et DAY n'ont pas été faites
- le timer ne se réinitialise pas après chaque commande entrée
- pas de système de détection d'erreur (commande introuvable/impossible d'attribuer cette valeur)

Documentation utilisateur :

1. Les différents modes

Le système comporte quatre modes différents :

Mode standard

Le système récupère à intervalle régulier la valeur des capteurs (l'intervalle entre 2 mesures est de 10 minutes par défaut). L'ensemble des mesures est enregistré sur une seule ligne horodatée et ces données sont enregistrés dans la carte SD.

Mode configuration

Ce mode permet de configurer le système grâce à une interaction depuis une console sur l'interface série du microcontrôleur (UART). Depuis l'interface on pourra taper des commandes de configuration pour modifier les valeurs des paramètres d'exécution.

Mode maintenance

Les données ne sont plus écrites sur la carte SD mais peuvent être consultées en direct depuis le port série. La carte SD peut être retirée et replacée en toute sécurité.

Mode économie

L'acquisition des données n'est plus effectuée qu'une mesure sur deux et le temps entre 2 mesures est multiplié par 2 tant que le système est dans ce mode.

2. Utilisation

Une fois la carte Arduino alimentée, il ne vous reste plus qu'à démarrer la station. Pour faire cela, il faut savoir dans quel mode voulez-vous démarrer.

Le système peut être démarré soit dans le mode standard, soit dans le mode configuration.

Pour démarrer la station dans le mode standard, maintenez le bouton entre une et cinq secondes. Pour le mode configuration, maintenez le bouton plus de cinq secondes.

Le schéma ci-dessous représente les différents changements de modes et leurs activations.

Nous utilisons un seul bouton afin de remédier au problème d'interférence de ceux-ci.

- Le bouton rouge correspond à une pression de moins de 5 secondes.
- Le bouton vert correspond à une pression de plus de 5 secondes.

3. Configuration

Afin de configurer les différents paramètres du système, on utilise le mode configuration afin d'entrer des commandes dans l'interface série. Il vous suffit d'entrer le nom du paramètre à modifier, de valider la commande puis d'entrée la nouvelle valeur que vous voulez assigner à ce paramètre. Vous trouverez ci-dessous la liste des paramètres

modifiable.

Paramètre	Domaine de définition des valeurs	Valeur par défaut	Description	Exemple de commande
LUMIN	{0,1}	1	définition de l'activation (1) / désactivation (0) du capteur de luminosité	LUMIN=1
LUMIN_LOW	{0-1023	255	définition de la valeur en dessous de laquelle la luminosité est considérée comme "faible"	LUMIN_LOW=200
LUMIN_HIGH	{0-1023}	768	définition de la valeur au-dessus de laquelle la luminosité est considérée comme "forte"	LUMIN_HIGH=700
Les valeurs comprises entre LUMIN_LOW et LUMIN_HIGH sont considérées comme "moyennes"				
TEMP_AIR	{0,1}	1	définition de l'activation (1) / désactivation (0) du capteur de température de l'air	TEMP_AIR=0
MIN_TEMP_AIR	{-40-85}	-10	définition du seuil de température de l'air (en °C) en dessous duquel le capteur se mettra en erreur.	MIN_TEMP_AIR=-5
MAX_TEMP_AIR	{-40-85}	60	définition du seuil de température de l'air (en °C) au-dessus duquel le capteur se mettra en erreur.	MAX_TEMP_AIR=30
HYGR	{0,1}	1	définition de l'activation (1) / désactivation (0) du capteur d'hygrométrie	HYGR=1
HYGR_MINT	{-40-85}	0	définition de la température en dessous de laquelle les mesures d'hygrométrie ne seront pas prises en compte.	HYGR_MINT=0
HYGR_MAXT	{-40-85}	50	définition de la température au-dessus de laquelle les mesures d'hygrométrie ne seront pas prises en compte.	HYGR_MAXT=50

PRESSURE	{0,1}	1	définition de l'activation (1) / désactivation (0) du capteur de pression atmosphérique.	PRESSURE=0
PRESSURE_MIN	{300-1100}	850	définition du seuil de pression atmosphérique (en HPa) en dessous duquel le capteur se mettra en erreur.	PRESSURE_MIN=450
PRESSURE_MAX	{300-1100}	1080	définition du seuil de pression atmosphérique (en HPa) au-dessus duquel le capteur se mettra en erreur.	PRESSURE_MAX=1030

La date et l'heure pourront être définis pour le module RTC grâce aux commandes suivantes :

- CLOCK → Configuration de l'heure du jour au format HEURE {0-23}, MINUTE {0-59}, SECONDE {0-59}.
- DATE → Configuration de la date du jour au format MOIS {1-12}, JOUR {1-31}, ANNEE {2000-2099}.
- DAY → Configuration du jour de la semaine {MON, TUE, WED, THU, FRI, SAT, SUN}.
- LOG_INTERVALL=10 → Définition de l'intervalle entre 2 mesures, 10 minutes par défaut.
- FILE_MAX_SIZE=4096 → Définition de la taille maximale (en octets) d'un fichier de log, une taille de 4 ko provoque son archivage.
- RESET → Rénitialisation de l'ensemble des paramètres à leurs valeurs par défaut.
- VERSION → Affiche la version du programme et un numéro de lot (permet de tracer la production).

Pour les différents capteurs :

TIMEOUT=30 → durée (en secondes) au bout de laquellel'acquisition des données d'un capteur est abandonnée. Après 2 mesures en timeout, le capteur est signalé en erreur.

Conclusion :

Malheureusement, par un énorme manque de temps, ce livrable n'est qu'une ébauche de ce qu'il devrait être. Il manque également le programme Config par sa complexité spatiale mais aussi sémantique.