



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

Samuel Mertenat
Internet et Communication - Classe T2f
Kit ARDUINO #134

TABLE DES MATIERES

1 Historique du document.....	4
2 Introduction.....	6
3 Buts et Objectifs du Projet	6
4 Analyse	7
4.1 Arduino UNO	7
4.2 Arduino IDE.....	7
4.3 Fonctionnement du breadboard	8
4.4 Langage « Arduino »	9
4.5 Courant électrique.....	12
4.6 Capteurs	15
4.7 La mémoire.....	17
4.8 Application Controller Interface (ACI)	18
4.9 nRFgo Studio.....	23
4.10 Sniffer et Wireshark.....	28
5 Conception.....	34
5.1 Blink	34
5.2 Logging.....	34
5.3 Mémoire libre	34
5.4 Schéma de connexion des modules	35
5.5 Utilisation des capteurs	36
5.6 Watchdog Timer	37
5.7 Bluetooth LE – nRF8001	37
5.8 Classe C++ spécifique à la station météo.....	37
5.9 Classe C++ « Button »	40
5.10 Classe C++ « Notification »	41
5.11 Classe C++ « Measurements ».....	42
6 Réalisation	45
6.1 Blink	45
6.2 Logging.....	45
6.3 Mémoire libre	47
6.4 Utilisation des capteurs	47
6.5 Bluetooth LE – nRF8001	48
6.6 Classe C++ spécifique à la station météo.....	50
6.7 Classe C++ « Button »	53
6.8 Classe C++ « Notification »	55
6.9 Classe C++ « Measurements »	57
7 Tests et validation	60
7.1 TP01.....	60
7.2 TP02.....	62
7.3 TP03.....	65
7.4 TP04.....	68
7.5 TP05.....	70
7.6 TP06.....	73

8	Problèmes rencontrés & Solutions	77
9	Acquis	77
10	Perspectives	78
11	Conclusion	79
12	Annexe	79
13	Références	80

INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

1 HISTORIQUE DU DOCUMENT

13/03/2015 : Rendu du TP01

- Installation de l'IDE et création d'un premier projet : « IoT »
- Développement d'un programme permettant de faire clignoter une LED et faisant appel au « Serial » afin d'afficher un compteur sur le terminal
- Développement d'un module de « tracing », permettant d'afficher des informations relatives à des erreurs, à du débogage ou simplement des informations
- Analyse du fonctionnement de la mémoire et création d'une méthode permettant d'afficher la quantité de mémoire Ram disponible

25/03/2015 : Rendu du TP02

- Prise en compte des corrections à réaliser suite à la correction du TP01 (4.5.4, 4.5 et 0)
- Analyse du fonctionnement d'un breadboard (4.3), d'un diviseur de tension (4.5.6), d'une résistance « pullup » (4.5.5), d'un capteur DHT11 (4.6.1) et d'une photorésistance (4.6.2)
- Réalisation du schéma complet des connexions des différents modules à l'Arduino Uno (logiciel « Fritzing » ; 5.4)
- Conceptualisation de l'utilisation des capteurs (5.5) et du Watchdog Timer (5.6)
- Développement de la partie relative aux capteurs (6.4) et validation du fonctionnement du programme (7.2)

17/04/2015 : Rendu du TP03

- Prise en compte des corrections à effectuer (4.6.2)
- Analyse du fonctionnement de l'ACI (0)
- Conception de l'utilisation de la puce nRF8001 (5.7)
- Réalisation de la partie relative à l'utilisation du nRF8001 (6.5)
- Tests et validation (7.3)

23/05/2015 : Rendu du TP04 :

- Prise en compte des corrections à effectuer (5.7)
- Analyse du logiciel « nRFgo studio » et des paquets émis en mode BROADCAST et CONNECT (4.9 et 4.10)
- Conception de la classe « nRF8001MeteoStation » (5.8)
- Réalisation de la classe « nRF8001MeteoStation » (6.6)
- Tests et validation (7.4)

28/05/2015 : Rendu du TP05 :

- Mise à jour de la présentation du projet (3) et des librairies utilisées (4.2)
- Modification du schéma des connexions des différents modules (5.4)
- Ajout des nouvelles caractéristiques nécessaires à notre projet (4.9)
- Conception des classes « Button » et « Notification » (5.9 et 5.10)
- Réalisation des classes « Button » et « Notification » (6.7 et 6.8)
- Tests et validation (7.5)

15/06/2015 : Rendu du TP06 :

- Corrections : ajout d'une remarque suite à l'analyse des données émises par la connectique Bluetooth (4.10.2)
- Conception de la classe « Measurements » (5.11)
- Réalisation de la classe « Measurements » (6.9)
- Tests et validation (7.6)

2 INTRODUCTION

Dans le cadre de ce projet, nous allons essayer de mettre en place une station météo connectée, de laquelle nous pourrons consulter les données depuis un Smartphone, via le Bluetooth Low Energy. La station météo sera basée sur un Arduino Uno, auquel nous associerons par la suite, capteurs et actuateurs, tels qu'un capteur de température et d'humidité ou une photorésistance. Le projet se déroulera sur plusieurs sessions de travaux pratiques durant lesquelles nous implémenterons, au fur et à mesure, les différentes fonctionnalités et concepts vus en cours.

3 BUTS ET OBJECTIFS DU PROJET

Ce projet a pour but de mettre en pratique la matière étudiée durant le cours « Internet des objets » en développant, petit à petit, une station météo.

Plus précisément :

- Mise en œuvre et utilisation d'une plateforme de prototypage Arduino
- Développement d'une station météo connectée
- Assimilation d'éléments particuliers et ciblés
 - Programmation en C/C++
 - Capteurs et actuateurs simples
 - Communication Bluetooth Low Energy

D'autre part, suite à l'ajout de nouveaux composants, la station météo est à présent caractérisées par / capable de :

- Relever et transmettre des mesures de température, d'humidité et de pression
- Fonctionner en tant que « thermostat », dont les températures minimale et maximale peuvent être définies à l'aide du Smartphone
- Notifier à l'utilisateur la température courante à l'aide de couleurs : bleu (inf. au thermostat), vert ou rouge (sup. au thermostat)
- Permettre l'activation / désactivation de notifications sonores (buzzer)

4 ANALYSE

4.1 ARDUINO UNO

L'Arduino UNO est une plateforme de développement très bon marché pour débuter à bricoler avec de l'électronique ou de l'automatisation. Son IDE est multiplateformes et l'Arduino ne nécessite, au minimum, que d'être branché à un port USB pour fonctionner.

Caractéristiques :

• Micro contrôleur :	ATmega328
• Tension d'alimentation interne :	5V
• Tension d'alimentation (recommandée) :	7 à 12V, limites : 6 à 20 V
• Entrées / sorties numériques :	14 dont 6 sorties PWM
• Entrées analogiques :	6
• Courant max par broches E / S :	40 mA
• Courant max sur sortie 3,3V :	50mA
• Mémoire Flash :	32 KB (bootloader 0.5 KB)
• Mémoire SRAM :	2 KB
• Mémoire EEPROM :	1 KB
• Fréquence d'horloge :	16 MHz
• Dimensions :	68.6mm x 53.3mm
• IDE de développement :	Arduino

L'arduino Uno est donc un bon compris pour réaliser notre station météo. En effet, la plateforme est très répandue et de nombreux capteurs et actuateurs sont disponibles sur le marché. De plus, il ne consomme que peu d'énergie, ce qui en fait un bon candidat pour un système énergétiquement autonome. Cependant, la mémoire embarquée étant assez faible, il faudra être attentif lors de la partie conception et réalisation.

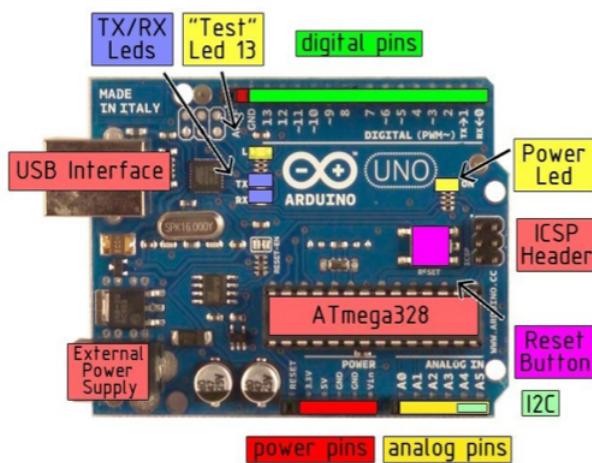


Figure 1: Carte Arduino Uno

4.2 ARDUINO IDE

Le logiciel de développement pour l'Arduino est disponible à l'adresse « <http://arduino.cc/en/Main/Software> » (version 1.6, Mac OS X édition).

L'interface est assez sommaire et permet de prendre rapidement le programme en mains.

Pour créer un nouveau projet, il suffit de cliquer sur « File > New » ; pour y ajouter un fichier : « Flèche vers le bas » (en haut à gauche) puis « New Tab ».

La compilation du code et le téléversement de celui-ci sur l'Arduino se font avec les boutons indiqués sur la capture ci-dessus.

Des librairies tierces peuvent être utilisées et doivent être installées dans le dossier « Arduino/librairies » (par simple copier / coller).

Librairie(s) installée(s) :

- DHT-sensor_library (Adafruit) : <https://github.com/adafruit/DHT-sensor-library>
- BLE (Semiconductor) : <https://github.com/NordicSemiconductor/ble-sdk-arduino>
- BMP085 : <https://github.com/adafruit/Adafruit-BMP085-Library>
- Timer : <https://github.com/JChristensen/Timer>

4.3 FONCTIONNEMENT DU BREADBOARD

Solderless Breadboards

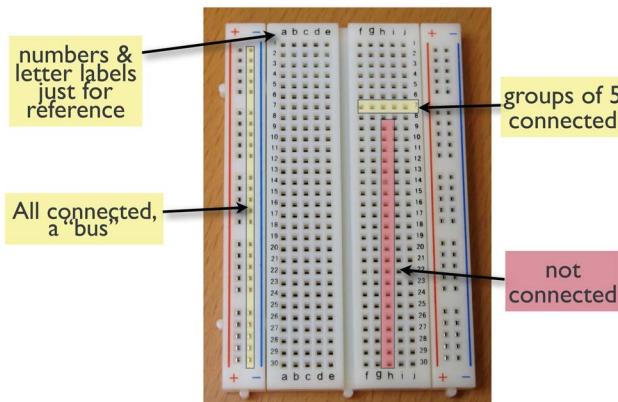


Figure 3: Planche à pain ou breadboard
(<http://yourduino.com/Photos/BreadBoard-1.jpg>)

une séparation au centre.

L'image ci-dessous permet de bien résumer ce principe de fonctionnement. Pour disposer d'une idée plus précise d'un circuit électronique, nous pouvons observer le schéma réalisé au point 5.4 de ce présent rapport, où l'on peut observer un Arduino Uno connecté à une

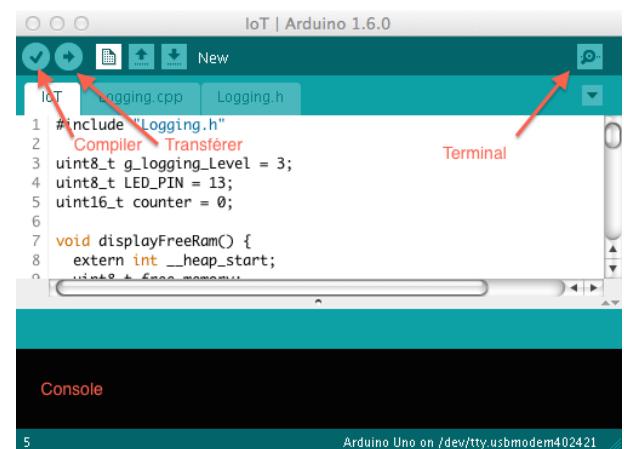


Figure 2: Interface de l'Arduino IDE

Une planche à pain ou breadboard, en anglais, est utilisé à des fins de prototypage et permet, sans soudure, de réaliser des circuits électroniques. L'avantage de ce système est d'être totalement réutilisable.

Son fonctionnement est assez simple, les trous des deux premières et les deux dernières colonnes sont reliés en bus, quant aux colonnes du milieu, les trois sont reliés en bus par ligne, avec

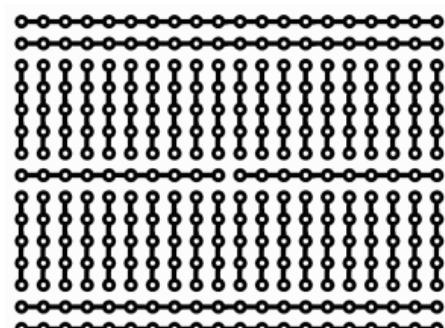


Figure 4: Fonctionnement des liaisons
(<http://upload.wikimedia.org/wikipedia/commons/7/75/Breadboard-144dpi.gif>)

photorésistance, à un capteur de température / humidité et à un module Bluetooth Low Energy.

4.4 LANGAGE « ARDUINO »

4.4.1 LE CODE MINIMAL

```
void setup()      //fonction d'initialisation de la carte
{
    //contenu de l'initialisation
}

void loop()       //fonction principale, elle se répète (s'exécute) à l'infini
{
    //contenu de votre programme
}
```

4.4.2 LES VARIABLES

Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32 768 à +32 767	16 bits	2 octets
long	entier	-2 147 483 648 à +2 147 483 647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octets
float	décimale	-3.4 x 10 ^{38} à +3.4 x 10 ^{38}	32 bits	4 octets
double	décimale	-3.4 x 10 ^{38} à +3.4 x 10 ^{38}	32 bits	4 octets

Voici le tableau des types non signés, on repère ces types par le mot `unsigned` (de l'anglais : non-signé) qui les précède :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octets
unsigned int	entier non négatif	0 à 65 535	16 bits	2 octets
unsigned long	entier non négatif	0 à 4 294 967 295	32 bits	4 octets

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables.

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
byte	entier non négatif	0 à 255	8 bits	1 octets
word	entier non négatif	0 à 65535	16 bits	2 octets
boolean	entier non négatif	0 à 1	1 bits	1 octets

Exemples :

```
boolean variable = FALSE; // variable est fausse car elle vaut FALSE, du terme anglais "faux"
boolean variable = TRUE; // variable est vraie car elle vaut TRUE, du terme anglais "vrai"

int variable = 0; // variable est fausse car elle vaut 0
int variable = 1; // variable est vraie car elle vaut 1
int variable = 42; // variable est vraie car sa valeur est différente de 0

int variable = LOW; // variable est à l'état logique bas (= traduction de "low"), donc 0
int variable = HIGH; // variable est à l'état logique haut (= traduction de "high"), donc 1
```

4.4.3 LES TABLEAUX

```
float notes[20]; //on créer un tableau dont le contenu est vide, on sait simplement qu'il contiendra 20 nombres
float note[] = {0,0,0,0 /*, etc.*/ };

float note[] = {};

void setup()
{
    note[0] = 0;
    note[1] = 0;
    note[2] = 0;
    note[3] = 0;
    //...
}
```

4.4.4 LES OPERATIONS LOGIQUES

4.4.4.1 IF ... ELSE IF

```
int prix_voiture = 5500;

if(prix_voiture < 5000)
{
    //la condition est vraie, donc j'achète la voiture
}

else if(prix_voiture == 5500)
{
    //la condition est vraie, donc j'achète la voiture
}

else
{
    //la condition est fausse, donc je n'achète pas la voiture
}
```

4.4.4.2 SWITCH

```
int options_voiture = 0;

switch (options_voiture)
{
    case 0:
        //il n'y a pas d'options dans la voiture
        break;
    default:
        //retente ta chance ;)
        break;
}
```

4.4.4.3 CONDITION TERNAIRE

```
int prix_voiture = 5000;
int achat_voiture = FALSE;

achat_voiture= (prix_voiture == 5000) ? TRUE : FALSE;
```

4.4.4.4 WHILE

```
while(/* condition à tester */)
{
    //les instructions entre ces accolades sont répétées tant que la condition est vraie
}
```

4.4.4.5 DO WHILE

```
do
{
    //les instructions entre ces accolades sont répétées tant que la condition est vrai
}while(/* condition à tester */);
```

4.4.4.6 FOR

```
for(int compteur = 0; compteur < 5; compteur++)
{
    //code à exécuter
}
```

4.4.5 LES FONCTIONS

4.4.5.1 SANS PARAMETRE

```
void fonction()
{
    int var = 24;
    return var; //ne fonctionnera pas car la fonction est de type void
}
```

4.4.5.2 AVEC PARAMETRE(S)

```
int x = 64;
int y = 192;

void loop()
{
    maFonction(x, y);
}

int maFonction(int param1, int param2)
{
    int somme = 0;
    somme = param1 + param2;
    //somme = 64 + 192 = 255

    return somme;
}
```

4.4.6 L'UTILISATION DU « SERIAL »

Le « Serial » permet d'écrire et de recevoir des données via le terminal. Pour ce faire, il faut initialiser la « ligne » avec un « `Serial.begin(9600)` », 9600 étant la vitesse en bits par seconde.

Les données peuvent être ensuite lues et écrites avec les méthodes suivantes (cf. doc) : `print()`, `write()`, `read()`, etc.

```

uint8_t LED_PIN = 13;
uint16_t counter = 0;

// the setup function runs once when you press reset or power the board
void setup() {
    Serial.begin(9600);           // initiates the serial communication
    pinMode(LED_PIN, OUTPUT);    // sets the pin as output
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_PIN, HIGH); // switches on the led
    Serial.print("The led is switched on; Number of times: ");
    Serial.println(++counter, DEC);
    delay(1000);                // waits for 1000ms -> 1s
    digitalWrite(LED_PIN, LOW);  // switches off the led
    delay(1000);
}

```

Source : <http://arduino.cc/en/reference/serial>

4.4.7 DIVERS

- Récupérer le temps : millis()
- Réajuster une valeur selon un intervalle : map(value, fromLow, fromHigh, toLow, toHigh)

4.5 COURANT ELECTRIQUE

4.5.1 SENS DU COURANT

Le courant électrique se déplace selon un sens de circulation. Un générateur électrique, par exemple une pile, produit un courant. Et bien ce courant va circuler du pôle positif vers le pôle négatif de la pile, si et seulement si ces deux pôles sont reliés entre eux par un fil métallique ou un autre conducteur. Ceci, c'est le sens conventionnel du courant.

4.5.2 INTENSITE DU COURANT

On mesure la vitesse du courant, appelée intensité, en Ampères avec un Ampèremètre. En général, en électronique de faible puissance, on utilise principalement le milliampère (mA) et le micro-Ampère (μ A), mais jamais bien au-delà.

4.5.3 TENSION

Autant le courant se déplace, ou du moins est un déplacement de charges électriques, autant la tension est quelque chose de statique. Pour bien définir ce qu'est la tension, sachez qu'on la compare à la pression d'un fluide.

4.5.4 LOI D'OHM

$$U = R * I$$

Dans le cas d'une LED, on considère, en général, que l'intensité la traversant doit-être de 20 mA. Si on veut être rigoureux, il faut aller chercher cette valeur dans le datasheet.

On a donc $I = 20$ mA.

Ensuite, on prendra pour l'exemple une tension d'alimentation de 5V (en sortie de l'Arduino, par exemple) et une tension aux bornes de la LED de 1,2V en fonctionnement normal. On peut donc calculer la tension qui sera aux bornes de la résistance :

$$U_r = 5 - 1,2 = 3,8 \text{ V}$$

Enfin, on peut calculer la valeur de la résistance à utiliser :

$$\text{Soit : } R = U / I$$

$$R = 3,8 / 0,02$$

$$R = 190 \text{ Ohms}$$

(Corrections TP01 ; ajout d'un schéma¹)

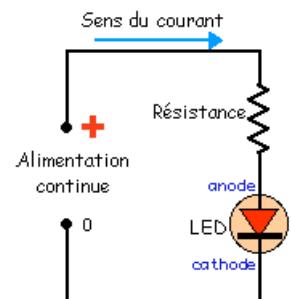


Figure 5: Schéma de la connexion

4.5.5 RESISTANCE « PULLUP »

Les résistances « pullup » sont fréquemment utilisées lors de l'utilisation de microcontrôleurs ; elles permettent de « corriger » un signal à un niveau logique High ou Low. Par exemple, si une broche est configurée en tant qu'entrée, que rien n'y est connecté et que le programme lit son état, il est difficile de prédire si le signal sera à un niveau High ou Low. Pour palier à ce problème, nous pouvons utiliser des résistances « pullup » ou « pulldown », qui permettent aussi d'utiliser moins de courant. Les résistances « pullup » sont les plus rependues et sont généralement utilisées avec des boutons ou des switches.

Calcul d'une résistance « pullup » :

En définissant arbitrairement la limite du courant à 1 mA lorsque le bouton est pressé, avec une tension V de 5 V, quelle doit-être la valeur de la résistance ?

La loi d'Ohm :

$$U = R * I$$

En appliquant au schéma, on obtient :

$$V_{cc} = (I \text{ à travers } R1) * R1$$

En isolant la résistance, on obtient :

$$R1 = \frac{V_{cc}}{I \text{ à travers } R1} = \frac{5V}{0.001A} = 5k \text{ Ohms}$$

Remarque : il ne faut pas utiliser une résistance trop élevée, ce qui diminuerait la tension, mais ralentirait le temps de réponse de la broche d'entrée pour détecter la variation².

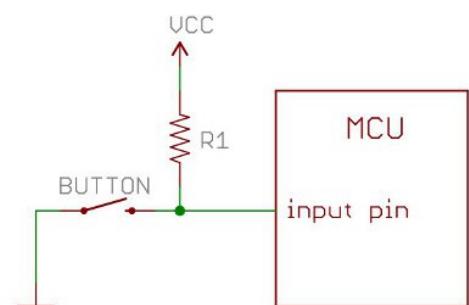


Figure 6: Schéma électrique d'un bouton-poussoir

¹ <http://www.ptitrain.com/electronique/tekno/thumbnails/04b.gif>

4.5.6 DIVISEUR DE TENSION

Un diviseur de tension, comme son nom l'indique, permet de réduire une tension en une plus faible. Ce mécanisme est possible en disposant deux résistances en série ; la tension de sortie sera donc une fraction de la tension d'entrée. Les applications sont variées : potentiomètre, photorésistance, etc.

Le circuit se compose donc de deux résistances et d'une tension d'entrée, en voici quelques exemples, sur la figure ci-dessous.

Afin de calculer la tension de sortie, la tension d'entrée et la valeur des deux résistances sont nécessaires.

La résistance R₁ se trouve toujours à au plus près de la tension d'entrée ; la résistance R₂ au plus près de la terre. La fraction de la tension d'entrée peut donc être mesurée entre ces deux résistances.

L'équation se présente ainsi :

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

Cette équation peut être simplifiée dans certaines situations. Si la valeur de la résistance R₁ est à égal à la résistance R₂, la tension de sortie est égale à la moitié de la tension d'entrée.

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} = V_{in} * \frac{R}{2R} = \frac{V_{in}}{2}$$

Si la valeur de la résistance R₂ est beaucoup plus élevée que la résistance R₁, la tension de sortie sera proche de la tension d'entrée.

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} \approx V_{in} * \frac{R_2}{R_2} = V_{in}$$

A l'inverse, si la valeur de la résistance R₂ est beaucoup plus faible que la résistance R₁, la tension passera majoritairement par la résistance R₁.

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} \approx V_{in} * \frac{0}{R_1} = 0$$

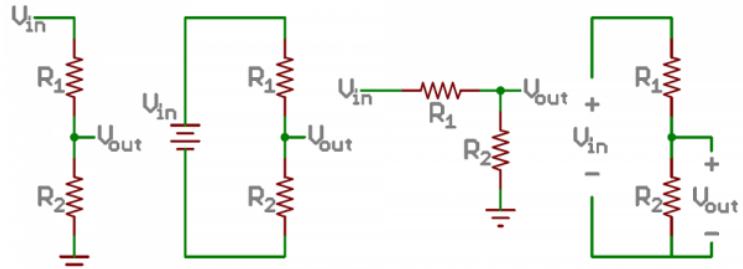


Figure 7 : Exemples de diviseur de tension

² <https://learn.sparkfun.com/tutorials/pull-up-resistors>

4.6 CAPTEURS

4.6.1 DHT11

Le capteur DHT11 permet de mesurer la température et l'humidité et est calibré d'usine. Son principal avantage est d'être très bon marché et relativement précis, comme nous pouvons le voir sur la figure ci-dessous.

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 °C	±5%RH	±2°C	1	4 Pin Single Row

Le capteur utilise une connexion « single-wire serial interface » pour communiquer et se synchroniser avec le microprocesseur ; une communication dure en principe, environ 4 ms. Les données sont codées sur 40 bits. Pour plus d'informations quant à l'implémentation du protocole de communication, nous pouvons nous référer à la datasheet du capteur, disponible à l'adresse : <http://www.micropik.com/PDF/dht11.pdf> (pages 5-9).

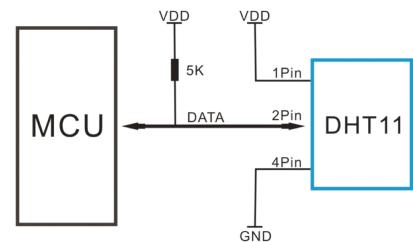


Figure 8: Branchement d'un capteur DHT11

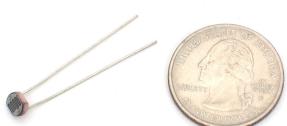
Son branchement est ais  , il ne nécessite que de trois connexions :

- Pin 1 : Vcc 5V
- Pin 2 : Data → résistance « pullup » d'environ 5k Ohms + pin digitale sur l'Arduino
- Pin 4 : GND

La valeur de la résistance « pullup » peut  tre facilement calcul e   l'aide de la formule pr sent e au point 4.5.5, en utilisant un courant de 1mA, correspondant   la consommation moyenne du capteur   ~5 V.

$$R1 = \frac{Vcc}{I \text{ \'a travers } R1} = \frac{5V}{0.001A} = 5k \text{ Ohms}$$

4.6.2 PHOTORESISTANCE



Une photor sistance est un composant  lectronique dont la r sistivit  varie en fonction de la lumi re ambiante, bien que qu'elle ne soit sensible de la m me mani re   toutes les longueurs d'onde (plus sensible   des couleurs comme le jaune / rouge qu'  des couleurs froides).

Figure 9: Une photor sistance

Sur le graphe ci-dessous, trouv  sur internet, nous pouvons constater l' volution de l'illumination en lux, en fonction de la r sistance.

Afin d'avoir une représentation un peu plus claire de ce que représente des « lux », voici quelques exemples tirés de notre quotidien :

- Nuit de plein lune : 0.5 lux
- Rue de nuit bien éclairée : 20-70 lux
- Appartement bien éclairé : 200-400 lux
- Extérieur par ciel couvert : 500-25000 lux
- Extérieur en plein soleil : 50000-100000 lux

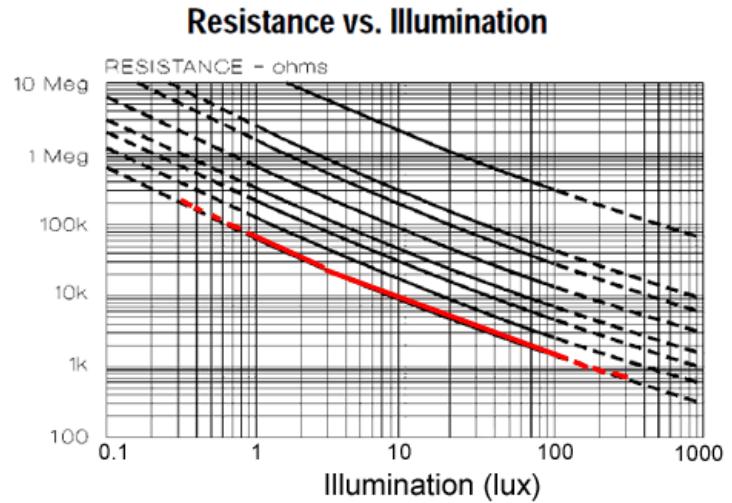
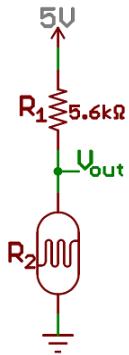


Figure 10: Graphique illustrant l'éclairement en fonction de la résistance
(<http://cdn.instructables.com/FNE/0LR9/FVS7L1OF/FNE0LR9FVS7L1OF.LARGE.gif>)



Pour mesurer la résistivité d'un capteur de ce type, il suffit de connecter une broche au 5 V et de relier la seconde à une résistance « pulldown », à une entrée analogique de l'Arduino et à la terre. Nous pouvons ensuite déterminer la résistance de la photorésistance à l'aide de l'Arduino, en fonction du courant mesuré.

$$V_o = V_{cc} \times \frac{R}{R + \text{Photorésistance}}$$

Le courant est donc inversement proportionnel à la résistance de la photorésistance et donc, aussi inversement proportionnel à la luminosité mesurée.

Les deux tableaux ci-dessous illustrent le courant mesuré, basé sur le capteur de photorésistance à 5 V et avec des résistances « pulldown » de 1k et de 10k Ohms.

Ambient light like...	Ambient light (lux)	Photocell resistance (Ω)	LDR + R (Ω)	Current thru LDR + R	Voltage across R
Dim hallway	0.1 lux	600K Ω	610 K Ω	0.008 mA	0.1 V
Moonlit night	1 lux	70 K Ω	80 K Ω	0.07 mA	0.6 V
Dark room	10 lux	10 K Ω	20 K Ω	0.25 mA	2.5 V
Dark overcast day / Bright room	100 lux	1.5 K Ω	11.5 K Ω	0.43 mA	4.3 V
Overcast day	1000 lux	300 Ω	10.03 K Ω	0.5 mA	5V

Figure 11: Lux mesurés avec 5 V et une résistance de 10k Ohms

Ambient light like...	Ambient light (lux)	Photocell resistance (Ω)	LDR + R (Ω)	Current thru LDR+R	Voltage across R
Moonlit night	1 lux	70 K Ω	71 K Ω	0.07 mA	0.1 V
Dark room	10 lux	10 K Ω	11 K Ω	0.45 mA	0.5 V
Dark overcast day / Bright room	100 lux	1.5 K Ω	2.5 K Ω	2 mA	2.0 V
Overcast day	1000 lux	300 Ω	1.3 K Ω	3.8 mA	3.8 V
Full daylight	10,000 lux	100 Ω	1.1 K Ω	4.5 mA	4.5 V

Figure 12: Lux mesurés avec 5 V et une résistance de 1k Ohms

4.7 LA MEMOIRE

L'Arduino ne disposant que de très peu de mémoire, il est important de l'utiliser avec parcimonie. Comme nous pouvons l'observer sur la figure ci-dessous, la mémoire RAM est divisée en 4 parties distinctes: « .data variables », « .bss variables », « heap » et « stack ». La mémoire à notre disposition, pour instancier des variables, débute à la partie « heap » (« __heap_start ») et se termine à la fin de la « stack » (« __brkval »).

(Corrections TP01) Le « sketch » ou programme réalisé avec l'IDE Arduino sera stocké dans la mémoire FLASH de l'Arduino, qui en comporte 32kb, lors du « téléversement ». Quant à la mémoire SRAM, de 2kb, elle permettra de stocker les variables utilisées tout au long du programme ; c'est cette quantité de mémoire qui peut être réduite en faisant varier le niveau de logging.

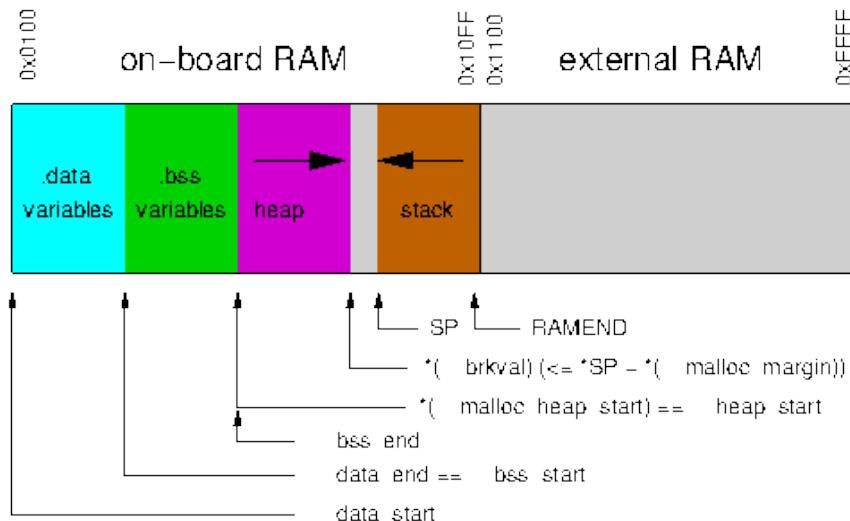


Figure 13: La mémoire

4.8 APPLICATION CONTROLLER INTERFACE (ACI)

Le ACI est une interface série, fonctionnant de manière bidirectionnelle, qui permet de gérer la puce nRF8001. L'ACI se place entre la couche application et les couches inférieures de la pile Bluetooth.

4.8.1 FONCTIONNEMENT

Le trafic d'information généré par l'ACI est bidirectionnel et le contrôle est exercé par le contrôleur de l'application (« application controller »). Ce dernier peut envoyer des commandes ACI au nRF8001 ; le nRF8001, peut quant à lui, envoyer des informations au contrôleur de l'application, de manière indépendante.

Les commandes envoyées par le contrôleur au nRF8001 peuvent être classifiées par :

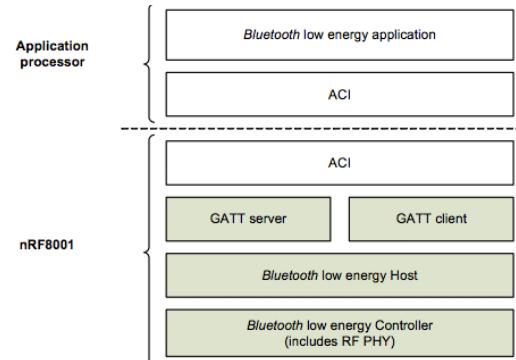


Figure 14: Pile du Bluetooth BLE

- Commandes système (« System commands ») : commandes utilisées pour la configuration et la gestion du mode opérationnel du nRF8001
- Commandes de données (« Data commands ») : commandes utilisées en état connecté, de façon « point-to-point » afin de transférer ou recevoir des données

Les informations transmises du nRF8001 à l'ACI sont appelées des événements (« events ») ; ils peuvent être liés au système ou aux données.

Sur la figure ci-dessous, nous pouvons observer 4 scénarios liés à l'utilisation de l'ACI :

1. System command – System event

Le contrôleur envoie une commande système (« System command ») au nRF8001, qui lui répond avec acquittement, sous la forme d'un événement (« System event »).

2. System event

Le nRF8001 envoie un événement (« event ») au contrôleur de l'application, déclenché par une condition prédéfinie.

3. Data command – Data event

Le contrôleur envoie une commande de données (« Data command ») afin de notifier un envoi ou une réception de données au nRF8001. Si la transaction réussit, les données sont envoyées sous la forme d'un événement (« event »).

4. Data event

Le nRF8001 envoie un événement (« event ») au contrôleur, déclenché par un transfert de données ou une condition prédéfinie relative au transfert de données

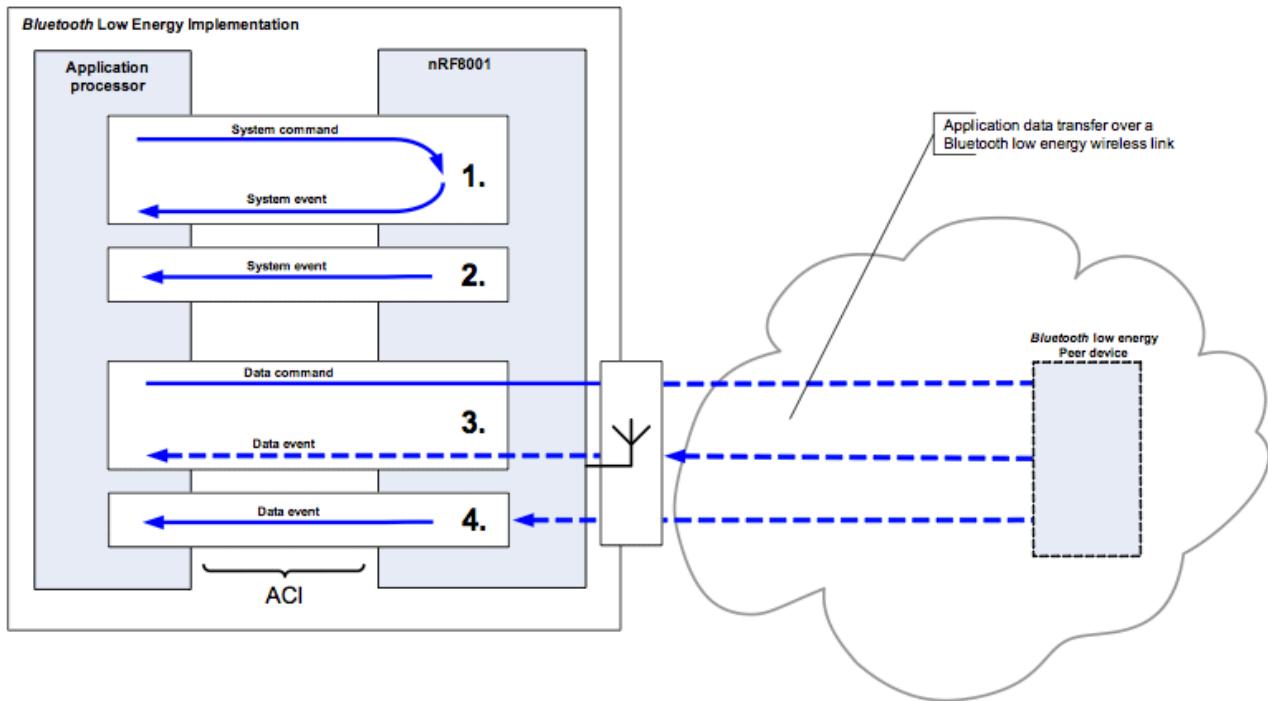


Figure 15: Principe de fonctionnement de l'ACI

4.8.1.1 STRUCTURE DES PAQUETS

Les différentes commandes et événements transitent par le biais de paquets. Chaque paquet est composé d'un entête de 2 octets (« Packet header ») et d'une charge utile (« Payload »), pouvant aller de 0 à 30 octets. L'ordre des données suit le format « Little Endian », soit que l'octet le moins significatif est transféré en premier.

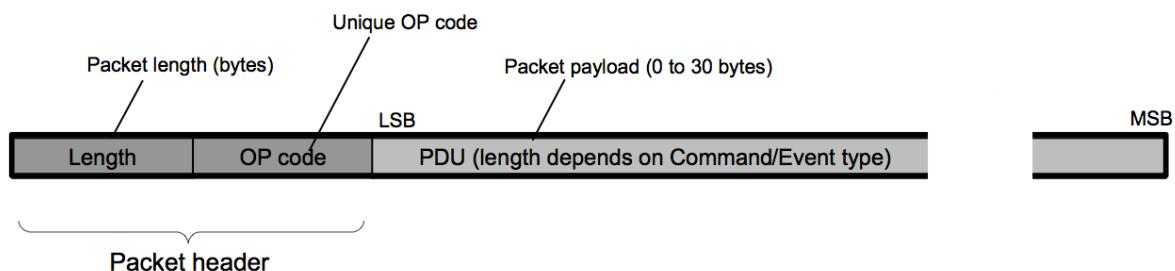


Figure 16: Structure d'un paquet ACI

Structure du paquet :

- Length : contient la taille du paquet, excluant la taille du champ « Length »
- OP code : contient l'identifiant unique de la commande / de l'événement
- PDU : contient la charge utile, dépend du type de paquet ACI à transmettre

L'ACI propose trois types de paquets, que sont les « System commands », les « Data commands » et les « Events ».

4.8.2 SYSTEM COMMANDS

Les « System commands » sont des commandes envoyées par l'ACI au nRF8001 ; elles permettent de contrôler la configuration, le mode d'utilisation et le comportement du nRF8001.

Quelques commandes :

- Test (0x01) : permet d'activer ou désactiver le mode « test » sur le nRF8001
- Echo (0x02) : permet de tester le bon fonctionnement de la couche de transport de l'ACI
- DtmCommand (0x03) : permet d'envoyer une commande « Direct Test » au nRF8001, en passant par l'interface de l'ACI
- Sleep (0x04) : permet d'activer le mode veille sur la puce radio (en veille jusqu'à une commande « Wakeup »)
- Wakeup (0x05) : permet de « réveiller » le nRF8001 après une période en veille

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Length	1	1	Packet length
Command	1	0x05	Wakeup

Figure 17: Exemple du paquet d'une commande ACI « Wakeup »

D'autres commandes existent ; pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf³ » aux pages 96-132.

4.8.3 DATA COMMANDS

Les « Data commands » sont des commandes envoyées par l'ACI au nRF8001 ; elles sont utilisées lors de transferts de données entre un périphérique et la puce radio. Ces commandes permettent d'initialiser le transfert de données entre la puce radio le périphérique couplé :

- Quand le nRF8001 agit en tant que serveur GATT, il peut :
 - Initialiser le transfert de données, stockées localement, vers le périphérique couplé
 - Recevoir des données du périphérique
- Quant le nRF8001 agit en tant que client GATT, il peut :
 - Envoyer des données vers le périphérique couplé
 - Demander la transmission de données de la part du périphérique
 - Demander la transmission de données de la part du périphérique sous la forme d'une indication (« Handle value indication ») ou d'une notification (« Handle value notification »).

³ https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/61913825

Quelques commandes :

- SetLocalData (0x0D) : permet de transmettre ou de recevoir des données lorsque le nRF8001 est en mode connecté, avec un périphérique
- SendData (0x15) : permet d'envoyer des données à périphérique couplé, par le biais d'un « service transmit pipe »
- SendDataAck (0x16) : permet de confirmer la réception de données d'un périphérique
- RequestData (0x17) : permet de demander des données à un périphérique, par le biais d'un « service receive pipe »
- SendDataNack (0x18) : permet d'envoyer un non-acquittement de réception de données de la part d'un périphérique

Message field/ parameter	Value size (bytes)	Data value	Description
Header			
Length	1	3	
Command	1	0x18	SendDataNack
Content			
PipeNumber	1		On which pipe the data is negatively acknowledged.
ErrorCode	1		Attribute protocol error code to be sent to the peer device.

Figure 18: Exemple du paquet d'une commande ACI « SendDataNack »

Pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf » aux pages 133-138.

4.8.4 EVENTS

Les « events » sont des messages envoyés par le nRF8001 à destination de l'ACI ; ils peuvent être de simples réponses ou des événements asynchrones, déclenchés par un facteur lié à un certain événement :

- Response events : permet l'acquittement d'une commande
- Asynchronous events : permet l'indication à l'ACI qu'une condition a été remplie. Par exemple, un « DisconectedEvent » est généré lorsque la connexion radio a été perdue. Ces événements n'ont pas une relation temporelle régulière ou prévisible avec l'ACI.

Quelques évènements :

- DeviceStartedEvent (0x81) : permet d'indiquer un « reset recovery » ou un changement d'état
- EchoEvent (0x82) : permet de retourner une copie de l'« Echo ACI message »
- HardwareErrorEvent (0x83) : permet de retourner des informations liées au débogage d'éventuelles erreurs hardware
- CommandResponseEvent (0x84) : permet de confirmer la réception ou l'exécution d'une commande ACI
- ConnectedEvent (0x85) : permet d'indiquer qu'une connexion a été établie avec un périphérique

D'autres évènements existent ; pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf⁴ » aux pages 139-157.

4.8.5 SERVICE PIPES

Le concept de « service pipes » est propre à l'ACI du nRF8001, et non pas au Bluetooth. Il permet de simplifier l'accès à un service caractéristique dans un client / serveur. Un « service pipe » peut être considéré comme un tuyau, où circulent des données de / à un client / serveur.

Les « service pipes » permettent de pointer vers des déclarations spécifiques dans un service ; par exemple, la déclaration caractéristique de la température dans un service d'un thermomètre. La programmation de la configuration des « service pipes » se fait à l'intérieur du nRF8001 et est fixe pour la durée de l'application. Le type et le nombre de tuyaux sont à définir selon l'application et les données y seront transmises lors de l'exécution du programme.

L'installation des « service pipes » définit :

- La direction des transferts de données : transmission / réception
- La localisation du serveur : données stockées sur le nRF8001 ou sur le périphérique
- La nécessité des acquittements
- L'automatisation des acquittements
- L'authentification de la connexion
- Broadcast : données peuvent être envoyées via « connectable advertisement packets » ou « un-connectable advertisement packets »

Chaque « service pipe » dispose d'un identifiant unique ; lorsque des données sont envoyées / reçues à / de un serveur, l'identifiant permet à la partie logiciel de faire la correspondance entre le GATT et les fonctionnalités fournies par le « service pipe ».

Sur la figure ci-dessous, nous pouvons observer comment différents « service pipes » peuvent être assignés à deux déclaration de caractéristique d'un service.

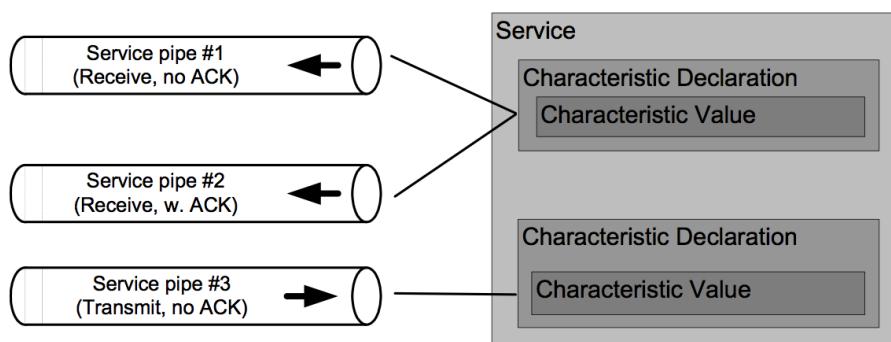


Figure 19: « Service pipes » assignés à un service

Pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf » aux pages 59-64.

⁴ https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/61913825

4.9 NRFGO STUDIO

nRFgo studio est un logiciel permettant de créer des clients (profiles) et des services GATT pour la puce nRF8001 et de générer les fichiers d'entêtes (« services.h ») correspondants, qui peuvent être ensuite importés dans un projet de développement.

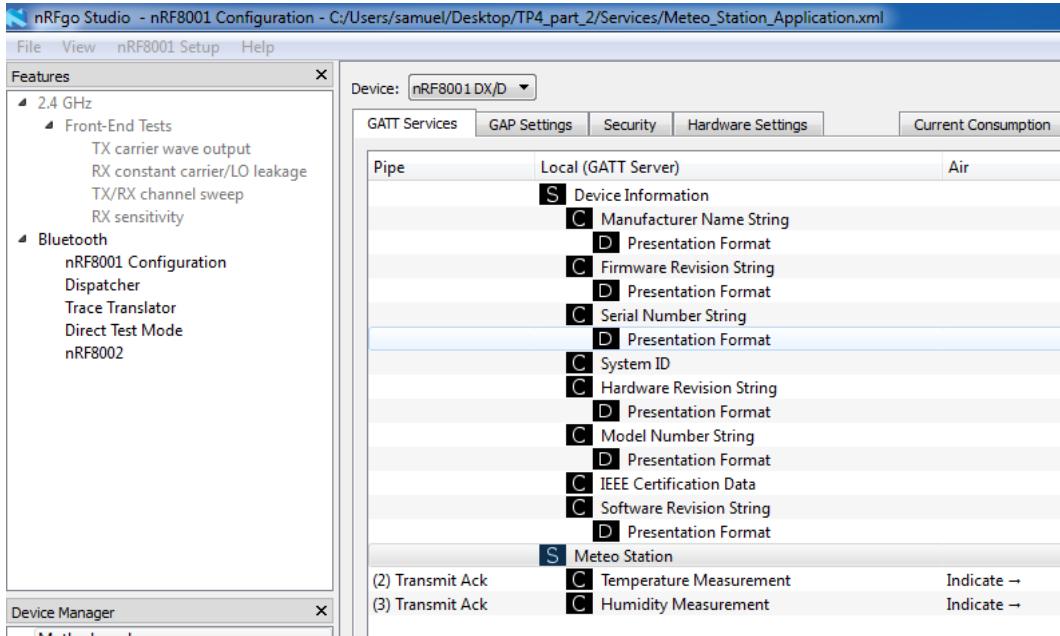


Figure 20: nRFgo Studio

Sur la figure ci-dessus, les services et les caractéristiques correspondent aux éléments utilisés par notre station météo. Seul le service « Meteo Station » a été créé par nos soins pour répondre aux exigences de notre projet, c'est-à-dire de pouvoir envoyer une mesure de température et d'humidité.

Services et caractéristiques correspondantes:

- Device Information (par défaut) :
 - Manufacturer Name
 - Firmware Revision
 - Serial Number
 - System ID
 - ...
- Meteo Station : notre service personnalisé
 - Thermostat Temperature
 - Temperature Measurement
 - Humidity Measurement
 - Pressure Measurement

GATT Services	GAP Settings	Security	Hardware Settings	Current Consumption
Pipe	Local (GATT Server)	Air	Re	
S	Device Information			
C	Manufacturer Name String			
D	Presentation Format			
C	Firmware Revision String			
D	Presentation Format			
C	Serial Number String			
D	Presentation Format			
C	System ID			
C	Hardware Revision String			
D	Presentation Format			
C	Model Number String			
D	Presentation Format			
C	IEEE Certification Data			
C	Software Revision String			
D	Presentation Format			
S	Meteo Station			
(2) Receive	C Thermostat Temperature	– Write Without Response		
(3) Transmit Ack	C Temperature Measurement	Indicate →		
(4) Transmit Ack	C Humidity Measurement	Indicate →		
(5) Transmit Ack	C Pressure Measurement	Indicate →		

Figure 21: Liste des services et caractéristiques

Le service « **Meteo Station** » dispose d'un UUID personnalisé, basé sur le service « Health Termometer⁵ » (0x1809). Dès lors, comme le service « Meteo Station » ne respecte pas les contraintes liées à ce service, il est nommé « Unknown Service » à l'intérieur de l'application « nRF Master Control Panel » (cf. Tests et validation > TP04).

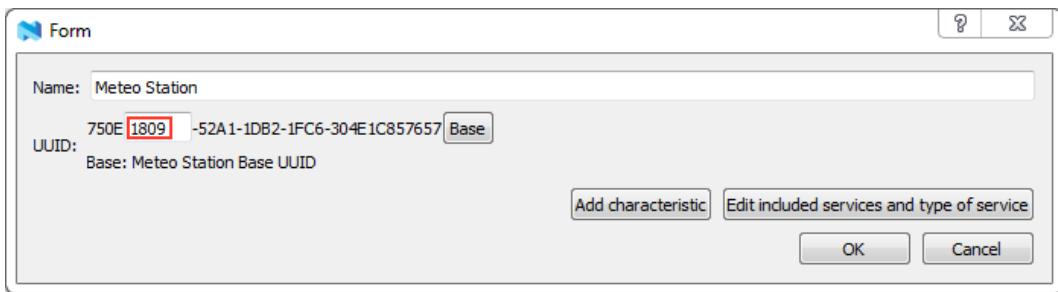


Figure 22: Service « Meteo Station »

Ce service, « Meteo Station » est caractérisé par deux caractéristiques : « Temperature Measurement » et « Humidity Measurement ».

La caractéristique « **Temperature Measurement** » est caractérisée par l'UUID 0x2A1C, qui est assigné à une mesure de température⁶. Cette caractéristique peut avoir une taille variable, de 5 à 13 bytes, en fonction l'inclusion ou non d'un type / timestamp (5 bytes : flags (1 byte) + données (4 bytes)).

Flags employés : 0b00000000 :

- Bit 0 → 0 : Celsius (1 : Fahrenheit)
- Bit 1 → 0 : Timestamp non présent
- Bit 2 → 0 : Temperature Type non présent
- Bits 3-7 → 0 : Pour des utilisations futures

Quant à la caractéristique « **Humidity Measurement** », elle dispose d'un UUID composé de la valeur « 0x2A6F », qui correspond à une mesure d'humidité⁷. Le format de la donnée doit être une valeur entière sur 16 bits, assigné en pourcentage. Ceci n'étant pas le cas, la valeur sera affichée sur le Smartphone en tant que valeur hexadécimal.

⁵ https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.health_thermometer.xml

⁶ https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.temperature_measurement.xml

⁷ https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.temperature_measurement.xml

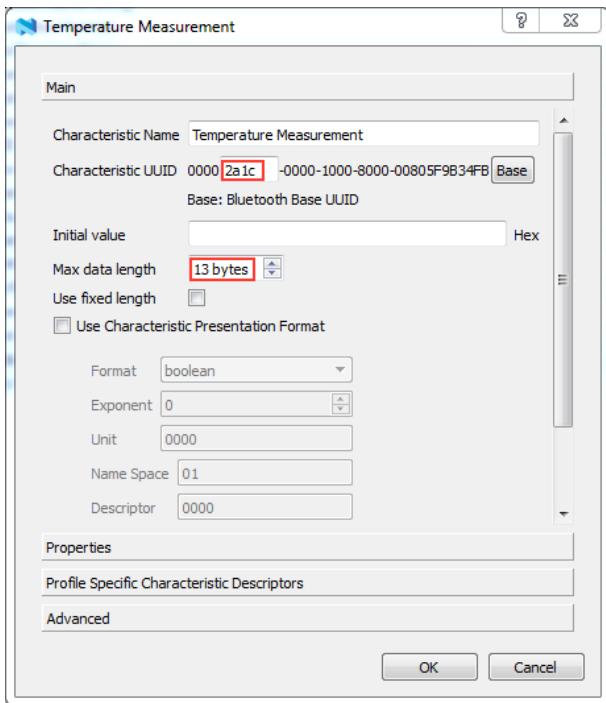


Figure 23: Caractéristique de température

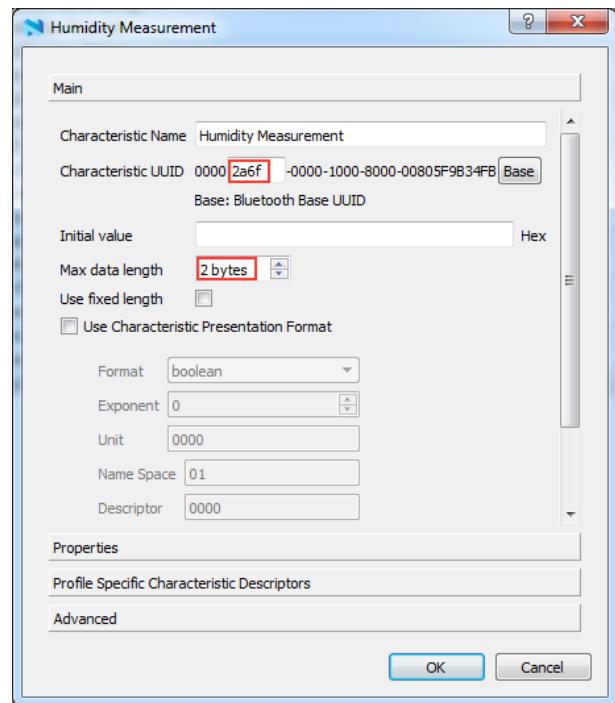


Figure 24: Caractéristique d'humidité

Pour que la caractéristique soit reconnue, il aurait fallu utiliser l'unité des pourcentages, mais nécessiterait une implémentation un peu plus complexe. Par manque de temps, la donnée sera envoyée simplement en tant que valeur entière.

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Humidity Information: Unit is in percent with a resolution of 0.01 percent Unit: <code>org.bluetooth.unit.percentage</code> Exponent: Decimal, -2	Mandatory	uint16	N/A	N/A	None

Figure 25: Caractéristique de l'humidité selon la norme BLE

Pour les besoins de notre projet, nous créons encore deux caractéristiques : une caractéristique pour transmettre la pression et une autre permettant l'envoi de données depuis le Smartphone vers la station météo.

Caractéristique « **Pressure Measurement** » :

- UUID : 2A6D (Pressure⁸)
- Taille : 4 bytes
- Indicate

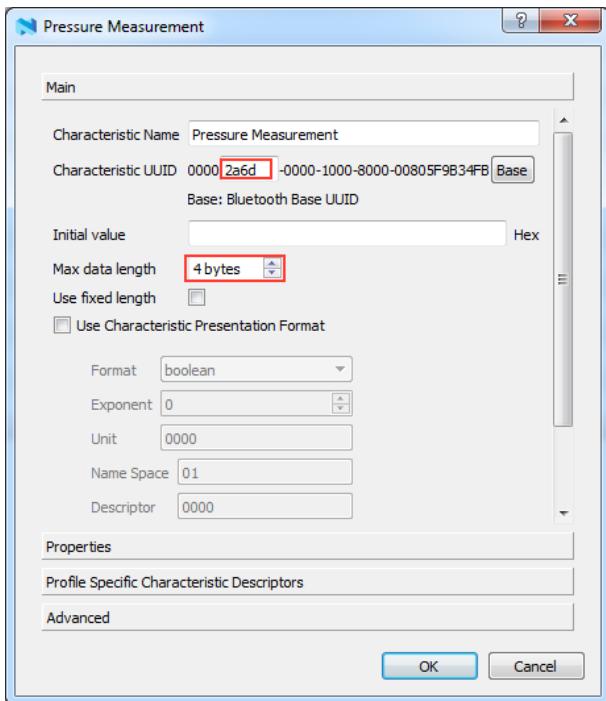


Figure 26: Caractéristique de pression

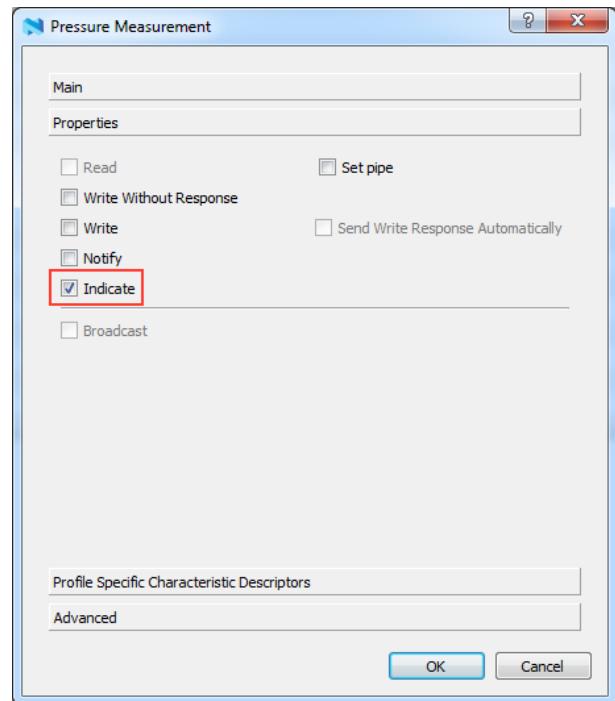


Figure 27: Caractéristique de pression

⁸ <https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.pressure.xml>

Caractéristique « Thermostat Temperature » :

- Valeurs par défaut : Add characteristic > UART RX
- UUID : 0x0002 base : Custom UART)
- Taille : 20 bytes (2 bytes seulement seront nécessaires)
- Write Without Response

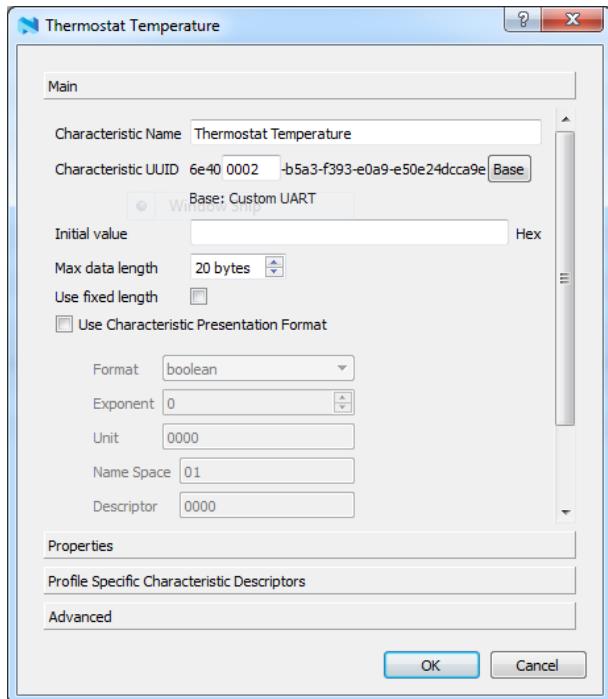


Figure 28: Caractéristique pour la réception de données

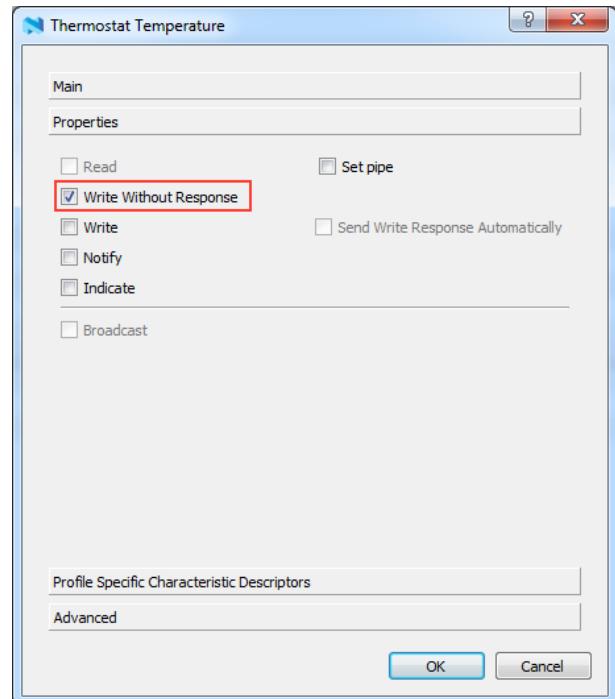


Figure 29: Caractéristique pour la réception de données

4.10 SNIFFER ET WIRESHARK

Afin de pouvoir capturer les paquets échangés à l'aide de la technologie Bluetooth, il est nécessaire d'utiliser un dongle Bluetooth, d'installer ses drivers, ainsi que d'utiliser un sniffer. Ce sniffer s'exécute en console, permet de lister les différents appareils à proximité, d'en sélectionner un (dans notre cas, le « 0 ») et de lancer une capture de paquet à l'aide de Wireshark (pression sur la touche « w »).

The screenshot shows a command-line interface for the Nordic Semiconductor Sniffer. It displays the following information:

- Sniffer ready and connected on COM4**
- Software version SUN rev. 1111**
- Firmware version SUN rev. 1111**
- Nordic Plugin version information unavailable**
- BTLE Plugin version SUN rev. 1111**
- Commands:**
 - l List the devices available for sniffing.
 - arrow keys Navigate the device list. Use ENTER to select.
 - [#] or ENTER Select a device to sniff from list.
 - e Like ENTER, but sniffer will only follow advertisements.
 - w Start Wireshark, the primary viewer for the sniffer (highlighted in red)
 - x/q Exit
 - c Display filter: Nearest devices <RSSI > -50 dBm>.
 - v Display filter: Nearest devices <RSSI > -70 dBm>.
 - b Display filter: Nearest devices <RSSI > -90 dBm>.
 - a Remove display filter.
 - p Passkey entry
 - o OOB key entry
 - h Define new adv hop sequence.
 - s Get support
 - u Launch User Guide (pdf)
 - CTRL-R Re-program firmware onto board
- Available devices:**

#	public name	RSSI	device address
-> [X] 0	"IoT_Samuel"	-60 dBm	[e8:51:bf:97:8b:37] random
[] 1	""	-103 dBm	64:dd:46:52:7c:45 random
Sniffing device 0 – "IoT_Samuel"			

Figure 30: Fenêtre en console du sniffer

Afin d'observer le trafic émis par un périphérique Bluetooth, nous procédons à deux captures, dans les situations suivantes :

- En mode BROADCAST, en utilisant le fichier « _BLE_broadcast_TxPower.h »
- En mode CONNECT, en utilisant le fichier « services.h » généré spécialement pour la station météo (cf. nRFgo studio, 4.9)

4.10.1 MODE BROADCAST

Lorsque nous nous trouvons en mode BROADCAST (connexion non possible), uniquement des paquets de type « ADV_NONCONN_IND » sont émis. Ceux-ci sont émis par le broadcaster (nRF8001) à destination de tous les autres périphériques Bluetooth 4.0.

Filter: btle.advertising_address == E8:51:BF:97:8B:37							Expression...	Clear	Apply	Save
No.	Time	Source	Destination	MAC Src	MAC Dst	Protocol	Length	Info		
1	0.000000000	Slave	Master			LE LL	47	ADV_NONCONN_IND		
2	0.164826000	Slave	Master			LE LL	47	ADV_NONCONN_IND		
3	0.328613000	Slave	Master			LE LL	47	ADV_NONCONN_IND		

Figure 31: Aperçu des paquets échangés en mode BROADCAST

Lors nous nous trouvons en mode BROADCAST, la puce nRF8001 a le rôle de « Broadcaster », c'est-à-dire qu'il émet en permanence des paquets à destination de tous les autres périphériques (« Observer device »). La topologie du réseau ne comporte donc, en principe, qu'un Broadcaster et un ou plusieurs Observer device(s).

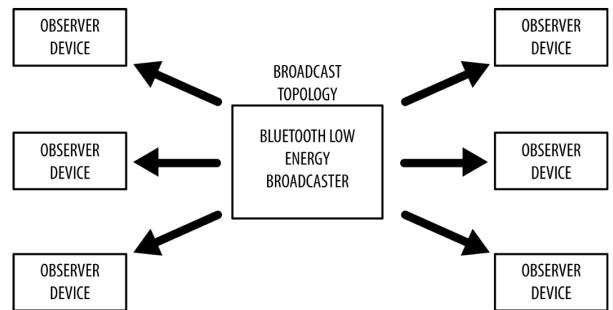


Figure 32: Topologie en mode BROADCAST

Comme nous pouvons l'observer sur le détail d'un paquet « ADV_NONCONN_IND », les données sont identiques à celles affichées sur le Smartphone. Un paquet est caractérisé par :

- Address type : 0x1542, généralement générée de manière aléatoire, pour des raisons de sécurité
- Access Address : 0x8e89bed6, adresse propre aux paquets d'advertising
- PDU type : ADV_NONCONN_IND
- Flags : indiquent que le périphérique ne supporte pas le Bluetooth BR/EDR
- The service list : Device name et Tx Power Level
- The device name : SAMeteo
- The TxPower Level characteristic : 0 [dBm]

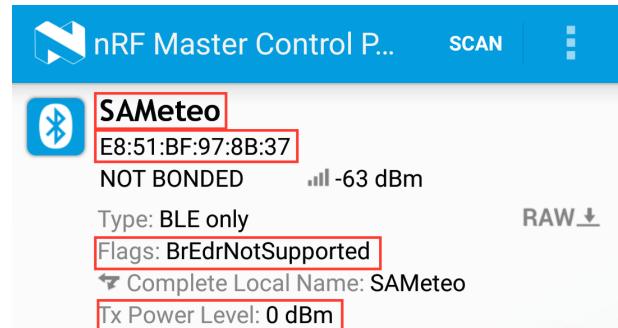


Figure 33: La puce nRF8001 en mode BROADCAST

Frame 1: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface 0
Nordic BLE sniffer meta

```

...
flags: 0x01
.... 0.. = encrypted: No
.... .0. = direction: Slave -> Master
.... ...1 = CRC: OK
channel: 38

Bluetooth Low Energy Link Layer
Access Address: 0x8e89bed6
Packet Header: 0x1542 (PDU Type: ADV_NONCONN_IND, TxAdd=false, RxAdd=false)
..00 .... = RFU: 0
..1... .... = Randomized Tx Address: True
..0.... .... = Reserved: False
.... 0010 = PDU Type: ADV_NONCONN_IND (0x02)
00.. .... = RFU: 0
..01 0101 = Length: 21
Advertising Address: e8:51:bf:97:8b:37 (e8:51:bf:97:8b:37)
Advertising Data
Flags
Length: 2
Type: Flags (0x01)
000.... = Reserved: 0x00
...0.... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x00)
.... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x00)
.... .1.. = BR/EDR Not Supported: true (0x01)
.... ..0. = LE General Discoverable Mode: false (0x00)
.... ...0 = LE Limited Discoverable Mode: false (0x00)
Device Name: SAMeteo
Length: 8
Type: Device Name (0x09)
Device Name: SAMeteo
Tx Power Level
Length: 2
Type: Tx Power Level (0x0a)
Power Level (dBm): 0
...

```

En observant maintenant le temps entre chaque paquet émis, nous pouvons constater qu'il correspond à la valeur spécifiée dans la méthode « begin(...) » de la classe « nRF8001Device ». En effet, les deux premiers paquets sont distants de 0.16 [s], ce qui correspond à 256 unités (0x100) de 0.625 [ms].

```
// configures the nRF8001 and starts advertising the configured services
// advTimeout is the duration during which advertising packets will be sent (in secs - 0 means infinite
// advertising)
// valid timeout range in second is from 1 to 16383
// advInterval is the time interval between advertising packets (in 0.625ms units)
// when broadcasting the valid range is from 0x0100 to 0x4000
// nbrOfEchoCommandsBeforeSetup is the number of echo commands issued before setup is done, useful for
// testing the SPI connectivity
// bool begin(uint16_t advTimeout, uint16_t advInterval, uint32_t nbrOfEchoCommandsBeforeSetup);
bool nrfIsStarted = nrf.begin(1, 0x0100, 0);
```

4.10.2 MODE CONNECT

Lorsque nous nous trouvons en mode CONNECT, il est possible de se connecter à la station météo et de récupérer la température et / ou l'humidité.

Pour initialiser une connexion, un « central device » récupère un paquet demandant une connexion (« connectable advertising packet ») provenant d'un « peripheral » et envoie ensuite une requête au périphérique afin qu'il établisse une connexion. Les deux périphériques peuvent, dès lors, transmettre des informations de façon bidirectionnelles.

Tant que personne n'est connecté, la puce nRF8001 émet des paquets « ADV_IND ».

Lorsque qu'un périphérique souhaite se connecter, il émet un paquet « CONNECT_REQ ». Dès que la connexion est achevée, la puce recommence à émettre des paquets « ADV_IND ».

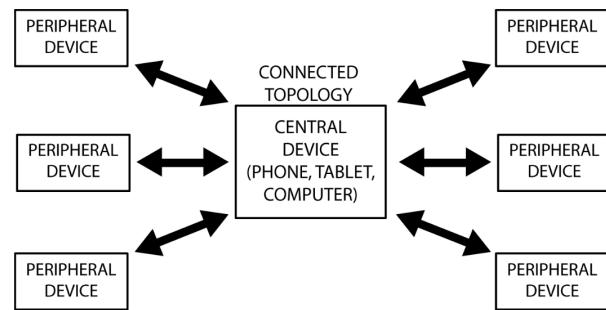


Figure 34: Topologie en mode CONNECT

Filter: btle		Expression... Clear Apply Save						
No.	Time	Source	Destination	MAC Src	MAC Dst	Protocol	Length	Info
67	4.583499000	Slave	Master			LE LL	62	ADV_IND
68	4.588738000	Slave	Master			LE LL	62	ADV_IND
69	4.592781000	slave	Master			LE LL	62	ADV_IND
70	4.595790000	Slave	Master			LE LL	60	CONNECT_REQ
71	4.598251000	Master	slave			LE LL	26	Empty PDU

Figure 35: Aperçu des paquets échangés en mode CONNECT, avant et après une connexion

Avant qu'une connexion soit établie, la puce nRF8001 émet des paquets « ADV_IND », au nombre de 3 et sur chaque canal d'advertising (37-38-39), toutes les 0.16 [s]. Sur le détail ci-dessous, nous pouvons observer que :

- Address type : 0x2440 généralement générée de manière aléatoire, pour des raisons de sécurité
- Access Address : 0x8e89bed6, adresse propre aux paquets d'advertising
- PDU type : ADV_IND

- Flags : indiquent que le périphérique ne supporte pas le Bluetooth BR/EDR
- The service list : Device name et notre service personnalisé, « Meteo Station », avec l'UUID « 5776851c4e30c61fb21da15209180e75 ». Cette valeur correspond à la valeur inverse définie dans le logiciel nRFgo studio pour le service « Meteo Station » (4.9)
- The device name : SAMeteo
- The TxPower Level characteristic : 0 [dBm]

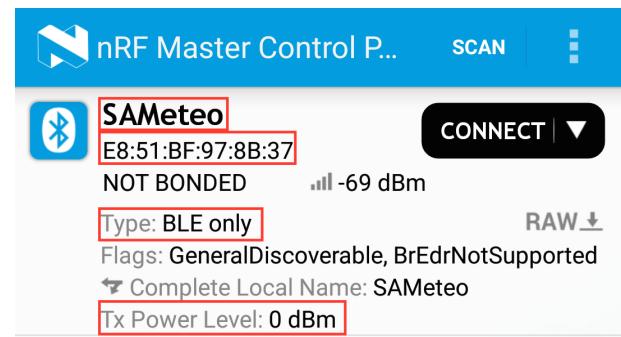


Figure 36: La puce nRF8001 en mode CONNECT

Frame 69: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
Nordic BLE sniffer meta

```
...
flags: 0x01
.... 0.. = encrypted: No
.... 0. = direction: Slave -> Master
.... .1 = CRC: OK
channel: 39
...
Bluetooth Low Energy Link Layer
Access Address: 0x8e89bed6
Packet Header: 0x2440 (PDU Type: ADV_IND, TxAdd=false, RxAdd=false)
..00 .... = RFU: 0
.1... .... = Randomized Tx Address: True
...0 .... = Reserved: False
.... 0000 = PDU Type: ADV_IND (0x00)
00... .... = RFU: 0
..10 0100 = Length: 36
Advertising Address: e8:51:bf:97:8b:37 (e8:51:bf:97:8b:37)
Advertising Data
Flags
Length: 2
Type: Flags (0x01)
000. .... = Reserved: 0x00
...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x00)
.... 0.. = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x00)
.... .1.. = BR/EDR Not Supported: true (0x01)
.... ..1. = LE General Discoverable Mode: true (0x01)
.... ...0 = LE Limited Discoverable Mode: false (0x00)
128-bit Service Class UUIDs
Length: 17
Type: 128-bit Service Class UUIDs (0x07)
Custom UUID: 5776851c4e30c61fb21da15209180e75
Device Name: SAMeteo
Length: 8
Type: Device Name (0x09)
Device Name: SAMeteo
```

Lorsque l'utilisateur initialise une connexion, en appuyant sur le bouton « CONNECT », un paquet « CONNECT_REQ » est émis et permet l'échange d'informations relatives aux acteurs de cet échange. Il est caractérisé principalement par :

- Address type : 0x22c5 généralement générée de manière aléatoire, pour des raisons de sécurité
- Access Address : 0x506563eb
- PDU type : CONNECT_REQ
- Interval : 39
- Latency : 0
- Timeout : 2000
- Channel Map : ffffffff1f, indique que les canaux de 0 à 36 sont disponibles pour la transmission de données ; les canaux 37-39 sont réservés pour réaliser de l'advertising

- Hop : 21

```
Frame 70: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Nordic BLE sniffer meta
...
flags: 0x01
.... 0... = encrypted: No
.... 0.. = direction: Slave -> Master
.... ..1 = CRC: OK
channel: 39
...
Bluetooth Low Energy Link Layer
Access Address: 0x8e89bed6
Packet Header: 0x22c5 (PDU Type: CONNECT_REQ, TxAdd=false, RxAdd=false)
..00 .... = RFU: 0
..1... .... = Randomized Tx Address: True
1... .... = Randomized Rx Address: True
.... 0101 = PDU Type: CONNECT_REQ (0x05)
00... .... = RFU: 0
..10 0010 = Length: 34
Initiator Address: 61:65:df:e1:3b:39 (61:65:df:e1:3b:39)
Advertising Address: e8:51:bf:97:8b:37 (e8:51:bf:97:8b:37)
Link Layer Data
Access Address: 0x506563eb
CRC Init: 0x943814
Window Size: 3
Window Offset: 31
Interval: 39
Latency: 0
Timeout: 2000
Channel Map: ffffffff1f
.... ...1 = RF Channel 1 (2404 MHz - Data - 0): True
...
.... 1.... = RF Channel 38 (2478 MHz - Data - 36): True
..0. .... = RF Channel 0 (2402 MHz - Reserved for Advertising - 37): False
..0. .... = RF Channel 12 (2426 MHz - Reserved for Advertising - 38): False
0.... .... = RF Channel 39 (2480 MHz - Reserved for Advertising - 39): False
1010 1... = Hop: 21
.... .100 = Sleep Clock Accuracy: 51 ppm to 75 ppm (4)
...
```

Suite à la connexion, de nouveaux types de paquets sont émis. Un paquet vide est de type « Empty PDU » ; quant aux données, elles sont transmises dans des paquets « Rcvd Handle value Indication » et acquittées avec des « Rcvd Handle value Confirmation ».

Tant que l'utilisateur n'a pas ouvert le pipe dédié à la température ou à l'humidité, il ne recevra pas de données (« Empty PDU »).

Filter: btle		Expression... Clear Apply Save						
No.	Time	Source	Destination	MAC Src	MAC Dst	Protocol	Length	Info
460	11.950167000	Master	Slave			LE LL	26	Empty PDU
461	11.953816000	Slave	Master			ATT	35	Rcvd Handle Value Indication, Handle: 0x0025
462	12.007264000	Master	Slave			ATT	31	Rcvd Handle Value Confirmation

Figure 37: Aperçu des paquets échangés lors de la réception et de l'acquittement de données

Un paquet « Value Indication » est caractérisé par :

- Address type : 0x0906, généralement générée de manière aléatoire, pour des raisons de sécurité
- Access Address : 0x506563eb, identique à celle attribuée au paquet « CONNECT_REQ »
- Channel : 9
- Next Expected Sequence Number : True → Attends un acquittement
- Bluetooth Attribute Protocol :
 - Opcode: Handle Value Indication (0x1d)
 - Value: 002d

```
Frame 461: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface 0
Nordic BLE sniffer meta
...
flags: 0x01
.... .0.. = encrypted: No
.... .0. = direction: Slave -> Master
.... .1 = CRC: OK
channel: 9
...
Bluetooth Low Energy Link Layer
Access Address: 0x506563eb
Data Header: 0x0006
    000. .... = RFU: 0
    ...0 .... = More Data: False
    .... 0... = Sequence Number: False
    .... 1.. = Next Expected Sequence Number: True
    .... ..10 = LLID: Start of an L2CAP message or a complete L2CAP message with no fragmentation
(0x02)
    000. .... = RFU: 0
    ...0 1001 = Length: 9
...
Bluetooth L2CAP Protocol
    Length: 5
    CID: Attribute Protocol (0x0004)
Bluetooth Attribute Protocol
    Opcode: Handle Value Indication (0x1d)
    Handle: 0x0025
    Value: 002d
```

Suite à l'envoi de la donnée (Indication), un acquittement est émis pour signaler la bonne réception des données. L'acquittement consiste à retourner la valeur « Handle Value Indication (0x1d) », à laquelle on ajoute un, par le biais d'un « Handle Value Confirmation (0x1e) ».

```
Frame 462: 31 bytes on wire (248 bits), 31 bytes captured (248 bits) on interface 0
Nordic BLE sniffer meta
...
flags: 0x03
.... .0.. = encrypted: No
.... .1. = direction: Master -> Slave
.... .1 = CRC: OK
channel: 21
...
Bluetooth Low Energy Link Layer
Access Address: 0x506563eb
Data Header: 0x050e
    000. .... = RFU: 0
    ...0 .... = More Data: False
    .... 1... = Sequence Number: True
    .... 1.. = Next Expected Sequence Number: True
    .... ..10 = LLID: Start of an L2CAP message or a complete L2CAP message with no fragmentation
(0x02)
    000. .... = RFU: 0
    ...0 0101 = Length: 5
...
Bluetooth L2CAP Protocol
    Length: 1
    CID: Attribute Protocol (0x0004)
Bluetooth Attribute Protocol
    Opcode: Handle Value Confirmation (0x1e)
```

La démonstration de ces deux dernières étapes est visible sur les figures du point 7.4.

(Corrections : TP06) : Le champ « Address Type » n'est pas généré de manière aléatoire, mais correspond aux données contenues dans l'en-tête du paquet.

5 CONCEPTION

5.1 BLINK

A l'étape 5 du premier TP, nous devions nous inspirer du code de l'exemple « Blink » (File > Examples > 01.Basics > Blink), faisant clignoter une la led 13 de l'Arduino toutes les deux secondes afin d'afficher à chaque allumage, un message sur la console, ainsi que le nombre de récurrence de cet événement. Pour réaliser ceci, il suffit de déclarer une nouvelle variable qui nous servira de compteur (« counter »), que l'on incrémentera à chaque tour de la « loop ». L'affichage se fait ensuite en utilisant le « Serial », en débutant par l'initialiser dans la fonction « setup() » (« Serial.begin(9600) ») et en affichant ensuite un message avec l'instruction « Serial.println() ».

5.2 LOGGING

A l'étape 6 du premier TP, nous devions réaliser un système nous permettant de « logger » notre code, car la plateforme Arduino n'en propose pas par défaut. Le principe consiste à afficher les logs sur la console, selon leur type (ERROR, INFO, DEBUG) et après quel temps d'exécution ils ont eu lieu. D'autre part, le niveau de « logging », qui peut prendre une valeur allant de 1 à 3, permet de spécifier la quantité de logs, et donc d'économiser de la mémoire (1 < 3).

Deux fichiers sont nécessaires à l'implémentation d'un tel système :

- Logging.cpp : Module contenant les méthodes « PrintHeader() » permettant d'afficher le moment où le message de log est affiché (hh-mm-ss), ainsi que le type de log (ERROR, INFO, DEBUG) et la méthode « PrintSerial », permettant d'utiliser une macro à deux paramètres et donc, de formater sur une seule ligne, toutes ces infos.
- Logging.h : Entête du fichier Logging.cpp contenant des instructions pour le préprocesseur afin de définir des macros pour les 3 types de logs. Le niveau de log définit ce que sera ou non compilé ; si le niveau de log est inférieur au niveau de log requis par les macros, celles-ci ne seront donc pas compilées.

5.3 MEMOIRE LIBRE

A l'étape 7 du premier TP, nous devions écrire une fonction permettant de calculer la quantité de mémoire libre. Bien que cette donnée soit affichée dans la console, il est intéressant de comprendre comment celle-ci est calculée.

Pour ce faire, il suffit d'instancier une variable (partie de la mémoire « stack »), d'en récupérer l'adresse et d'en soustraire l'adresse de la variable externe « __heap_start ».

5.4 SCHEMA DE ONNEXION DES MODULES

Sur le schéma ci-dessous, réalisé avec le logiciel « Fritzing », nous pouvons observer la manière dont les différents modules sont reliés à l'Arduino UNO.

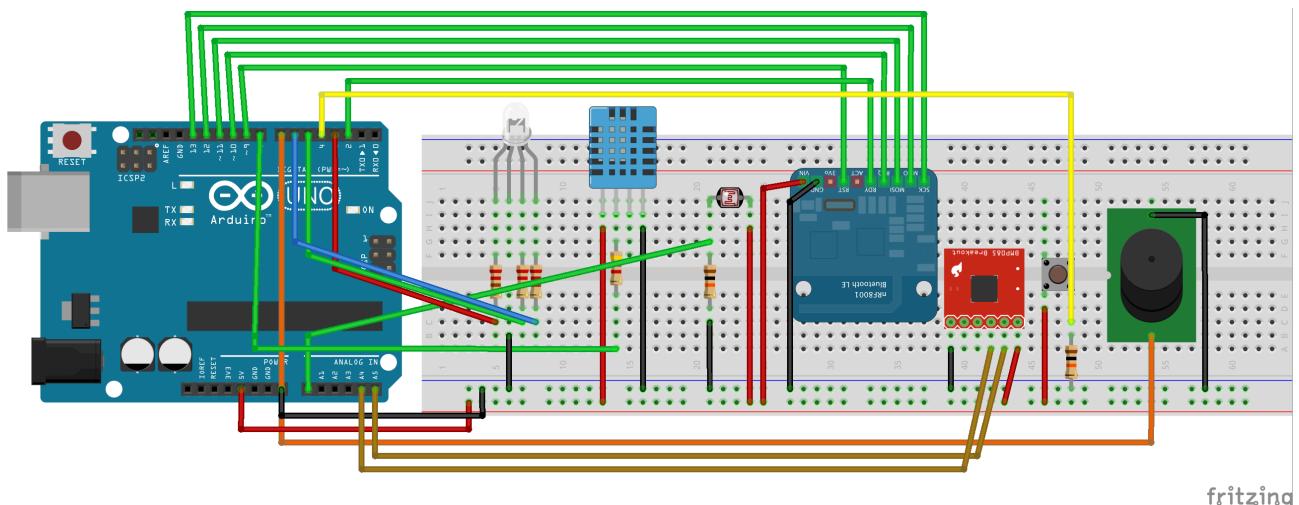


Figure 38: Schéma des connexions des différents modules

DHT11 :

- Pin 1 : 5 V
- Pin 2 : pin 8 (digitale) + résistance de 4.7 Ohms
- Pin 3 : terre

Photodiode :

- Pin 1 : pin A0 (analogique) + résistance de 10kOhms + terre
- Pin 2 : 5 V

nRF8001⁹ :

- Vin : 5 V
- GND : terre
- SCK : SPI clock, pin 13
- MISO : SPI MISO, pin 12
- MOSI : SPI MOSI, pin 11
- REQ : SPI Chip Select, pin 10
- RST : pin 9 (pour initialiser la carte)
- RDY : pin 2 (pour les interruptions)
- ACT : -

BMP085 :

- Vin : 5 V
- SDA : pin A4 (analogique, I2C)
- SCL : pin A5 (analogique, I2C)

⁹<https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout/hooking-everything-up>

- GND : terre

RGB led :

- Pin 1 : 3 (PWM, digitale) + résistance de 220 Ohms
- Pin 2 : terre
- Pin 3 : 5 (PWM, digitale) + résistance de 220 Ohms
- Pin 4 : 6 (PWM, digitale) + résistance de 220 Ohms

Bouton-poussoir :

- Pin 1 : 5 V
- Pin 2 : pin 4 (digitale) + résistance de 10kOhms + terre

Buzzer :

- Pin 1 : 7 (digitale)
- Pin 2 : terre

5.5 UTILISATION DES CAPTEURS

5.5.1 DHT11

Avant de pouvoir utiliser le capteur de température / humidité, il est nécessaire de copier / coller la librairie d'Adafruit dans l'IDE et d'inclure dans notre projet le fichier d'entêtes « DHT.h ». L'utilisation de cette librairie nous facilite grandement la tâche ; toutes les fonctions nécessaires à l'utilisation et à la prise de mesures sont, en effet, déjà implémentées.

Déroulement de l'utilisation du DHT11 :

- Importation de la librairie et référence vers le fichier « DHT.h »
- Définition du pin utilisé et déclaration du type de capteur utilisé (DHT11)
- Initialisation du capteur (dht.begin())
- Lecture du capteur (dht.readTemperature(), dht.readHumidity())
- (optionnel) Affichage de la mesure

5.5.2 PHOTORESISTANCE

L'utilisation de la photorésistance est très simple, il suffit de lire la valeur renvoyée par celle-ci, à l'aide de l'instruction « analogRead(A0) » (A0 étant l'entrée analogique où la photorésistance étant connectée).

Afin d'améliorer l'exactitude de la mesure de la luminosité, nous pouvons imaginer une phase de calibration au démarrage du programme, afin de calquer les mesures suivantes sur une valeur référence. D'autre part, il peut être intéressant de réduire l'intervalle des valeurs, en les disposant sur une échelle plus réduite. Nous utiliserons, dans notre cas, un intervalle de 0 à 100, à l'aide de la méthode « map() ».

5.6 WATCHDOG TIMER

Afin de palier au problème de la consommation énergétique de la station météo, il peut paraître nécessaire de réfléchir à comment économiser du courant et donc, si l'Arduino est connecté à une batterie, d'en prolonger la durée de vie.

Watchdog est un mécanisme incorporé au processeur AVR, qui permet de réinitialiser le processeur si une erreur survient. Il peut être aussi utilisé en tant que « timer », afin de réveiller le processeur d'un état de veille (« power-down sleep »). C'est donc pour cette application que nous allons l'utiliser, afin de mettre en veille et de réveiller l'Arduino entre les périodes de mesures. Plus de détails sur ce mécanisme peuvent être trouvés aux chapitres 9 et 10.8 du datasheet sur l'ATmega328P¹⁰.

Cette partie étant optionnel, la conception est à retrouver sur le site d'Adafruit : <https://learn.adafruit.com/low-power-wifi-datalogging/power-down-sleep> et le code utilisé, adapté pour notre station météo, provient du même site¹¹.

5.7 BLUETOOTH LE – NRF8001

Afin de pouvoir utiliser la puce radio nRF8001, diverses acquisitions seront à effectuer :

- Télécharger et importer la librairie BLE de Nordic Semiconductor¹²
- Télécharger et importer les fichiers « nRF8001Device.cpp / .h »¹³
- Télécharger et importer, selon de le mode de fonctionnement souhaité (broadcast ou connect), le fichiers « BLE_broadcast_TxPower » ou « BLE_connect_TxPower.h »

D'autre part, diverses modifications / ajouts seront à réaliser à l'intérieur du sketch :

- Inclure les fichiers d'entêtes relatifs à l'ACI et au bus SPI
- Définir des macros permettant de spécifier des arguments pour le constructeur de la classe « nRF8001Device »
- Définir des macros permettant d'utiliser ou non le code réalisé dans le TP précédent afin de réaliser des mesures de température, d'humidité et de luminosité
- Instancier un objet de la classe « nRF8001Device » et utiliser les différentes méthodes de la classe afin de l'initialiser et de l'utiliser

(Corrections TP4 : modification du point 3 du 1^{er} paragraphe)

5.8 CLASSE C++ SPECIFIQUE A LA STATION METEO

La création d'une classe C++ « nRF8001MeteoDevice » permet d'éteindre les fonctionnalités fournies par la classe « nRF8001Device ». Cette dernière permet d'utiliser la puce nRF8001 et s'occupe principalement de gérer l'état, les évènements et les commandes ACI. Afin d'implémenter des fonctions propres à notre station météo (envoi

¹⁰ <http://www.atmel.com/Images/doc8161.pdf>

¹¹ https://github.com/tdicola/Low_Power_Wifi_Datalogger/archive/master.zip

¹² <https://github.com/NordicSemiconductor/ble-sdk-arduino>

¹³ <http://cyberlearn.hes-so.ch/mod/folder/view.php?id=439257>

de données, réception d'acquittement), il est donc judicieux de créer une classe héritée de la classe « nRF9001Device ».

Cette classe définira :

- Une classe héritée de la classe « nRF8991Device »
- Un constructeur
- Une méthode permettant l'envoi de données
- Une méthode permettant la réception de l'acquittement des données envoyées
- Les structures destinées à l'envoi de la température et de l'humidité

L'envoi des données sera réalisé à partir de deux structures : une structure pour la température et une autre pour l'humidité.

Température:

- Flags (1 octet) : permet de définir l'unité utilisée, de préciser la présence d'un « timestamp », etc. Dans notre cas, les flags seront mis à zéro (température en Celsius, pas de « timestamp » ni de type).
- Mesure (4 octets) : permet de stocker la valeur de la température sous la forme d'un float.

Humidité :

- Mesure (2 octets) : permet de stocker la valeur de l'humidité, sous la forme d'un « int » limité à 16 bits. La valeur est stockée sous la forme d'un entier pour simplifier les choses ; l'utilisation du pourcentage est bien plus complexe à mettre en place, mais pourrait être facilement implémentée par le biais d'une application, sur le Smartphone.

5.8.1 CLASSE ET CONSTRUCTEUR

Avec le concept d'héritage, la classe « nRF8001MeteoStation » hérite de tous les attributs et fonctions de la classe « nRF8001Device ». Pour les besoins relatifs à l'envoi des données et à la réception des acquittements, il nous est nécessaire d'ajouter de nouveaux attributs à la classe :

- 2 attributs pour les structures de température et d'humidité (structures)
- 2 attributs pour les acquittements (booléens)
- 2 attributs pour stocker les précédentes données envoyées (température et humidité ; float).

Le constructeur de la classe appellera le constructeur de la classe mère (« nRF8001Device ») et initialisera les attributs membres (cités ci-dessus) de la manière suivante :

- Allocation de la mémoire pour les structures
- Affectation des variables relatives aux acquittements à faux (pas d'envoi, donc pas d'acquittement en attente)

- Affectation des variables relatives aux dernières données envoyées à la valeur maximale du type float (FLT_MAX).

5.8.2 ENVOI DES DONNEES

Le principe appliqué pour l'envoi des données est identique pour la température ou l'humidité. La démarche consiste à :

- Vérifier que la température / humidité à transmettre n'est pas identique à la température / humidité précédemment transmise ET vérifier que l'acquittement a été reçu pour le précédent envoi.
 - Si oui, vérifier que le « PIPE » correspondant à la caractéristique soit disponible (`ib_aci_is_pipe_available(...)`)
 - Si oui, assigner à la structure la donnée à transmettre (pour la température, renseigner aussi les flags)
 - Envoyer les données à l'aide de la fonction fournie par la librairie (`lib_aci_send_data`) et récupérer l'état, qui sera stocké dans la variable destinée à l'acquittement
 - Si la donnée a bien été transmise, assigner la donnée transmise à la variable relative à la dernière donnée envoyée
 - Retourner l'état de l'acquittement

5.8.3 RECEPTION DES ACQUITTEMENTS

Un acquittement est envoyé après chaque envoi ; pour déterminer la provenance de l'acquittement (Pipe), il est nécessaire de se renseigner sur les différentes structures déclarées dans le fichier « `aci_evts.h` » (librairie BLE). Les évènements ACI sont encapsulés dans une structure nommée « `aci_evt_t` », qui est composée de différents paramètres, sous la forme de structures. Un de ces paramètres se nomme « `data_ack` » (structure « `aci_evt_params_data_ack_t` ») et permet d'aller récupérer l'identifiant du pipe (attribut « `pipe_number` ») qui a émis un acquittement.

Cette fonction nécessitera donc qu'on lui passe en paramètre une structure de type « `aci_evt_t` » et sera appelée au sein de la fonction « `pollACI()` ».

La structure « `aci_evt_t` » :

```
/**
 * @struct aci_evt_t
 * @brief Encapsulates a generic ACI event
 */
typedef struct
{
    uint8_t len;
    aci_evt_opcode_t evt_opcode;
    union
    {
        aci_evt_params_device_started_t          device_started;
        aci_evt_params_echo_t                  echo;
        aci_evt_params_hw_error_t             hw_error;
        aci_evt_params_cmd_rsp_t            cmd_rsp;
        aci_evt_params_connected_t          connected;
        aci_evt_params_disconnected_t       disconnected;
        aci_evt_params_bond_status_t        bond_status;
        aci_evt_params_pipe_status_t        pipe_status;
        aci_evt_params_timing_t              timing;
        aci_evt_params_data_credit_t        data_credit;
        aci_evt_params_data_ack_t           data_ack; // highlighted in green
        aci_evt_params_data_received_t      data_received;
        aci_evt_params_pipe_error_t         pipe_error;
        aci_evt_params_display_passkey_t    display_passkey;
        aci_evt_params_key_request_t        key_request;
    } params;
} _aci_packed_ aci_evt_t;
```

La structure « aci_evt_params_data_ack_t » :

```
/**
 * @struct aci_evt_params_data_ack_t
 * @brief Structure for the ACI_EVT_DATA_ACK event return parameters
 */
typedef struct
{
    uint8_t pipe_number;
} _aci_packed_ aci_evt_params_data_ack_t;
```

5.9 CLASSE C++ « BUTTON »

La classe « Button » permettra d'instancier un objet « Button » permettant de détecter une simple pression ou une double pression sur un bouton-poussoir, dont la patte pourra être passée en paramètre au constructeur.

Cette classe est une version simplifiée et minimalisée de la librairie écrite par M. Matthias Hertel OneButton : <https://github.com/mathertel/OneButton>).

5.9.1 CLASSE ET CONSTRUCTEUR

La classe comporte 8 attributs :

- Le numéro de la patte sur laquelle le bouton est connecté
- Le nombre de [ms] (timeout) afin de définir s'il s'agit d'une pression simple / double
- Un pointeur vers la fonction appelée lors d'une simple pression
- Un pointeur vers la fonction appelée lors d'une double pression
- Le nombre de [ms] lors de la première pression (état 0)
- Le nombre de [ms] courant
- L'état dans lequel se trouve la machine d'état
- L'état logique du bouton-poussoir (LOW / HIGH)

Le constructeur effectue les tâches suivantes :

- Récupère la patte passée en paramètre et l'assigne à l'attribut dédié

- Définit le timeout à 800 [ms]. Cette valeur est optimale en raison des autres instructions exécutées dans la boucle principale (loop()).
- Initialise la machine d'état à 0
- Initialise l'état logique du bouton-poussoir à LOW
- Définit la patte du bouton-poussoir en tant qu'entrée

5.9.2 DISTINCTION DU TYPE DE PRESSION

Une méthode sera appelée l'intérieur de la boucle « loop() » et permettra de scruter l'état du bouton en continu.

Fonctionnement :

- Récupère l'état logique du bouton et le nombre de [ms] courant
- Si la machine d'états est dans l'état initial (=0) :
 - Si le bouton est pressé
 - La machine d'états passe à 1
 - Assigne le nombre de [ms] courant à l'attribut stockant le nombre de [ms] suite à une pression (m_start_time)
- Si la machine d'états est dans l'état 1 :
 - Si le bouton n'est pas pressé :
 - La machine d'états passe à 2
- Si la machine d'état est dans l'état 2
 - Si le temps courant est supérieur au temps lors de la première pression + timeout
 - Appelle la fonction attachée à un simple clic
 - La machine d'état repasse à l'état 0
 - Si le bouton est à nouveau pressé (HIGH)
 - La machine d'états passe à 3
- Si la machine d'états est dans l'état 3
 - Si le bouton n'est pas pressé
 - Appelle la fonction attachée à un double clic
 - La machine d'états repasse à l'état 0

5.10 CLASSE C++ « NOTIFICATION »

La classe « Notification» permettra d'instancier un objet « Notification » permettant de signaler un état quelconque à l'utilisateur à l'aide d'une LED et d'un buzzer.

5.10.1 CLASSE ET CONSTRUCTEUR

La classe comporte 8 attributs :

- Le numéro de la patte de la LED (rouge)
- Le numéro de la patte de la LED (vert)
- Le numéro de la patte de la LED (bleu)
- Le numéro de la patte du buzzer
- La durée de la notification

- L'intervalle entre deux notifications (par exemple, si on veut faire clignoter la LED)
- L'état du buzzer (activé / désactivé)

Une énumération, pour indiquer les différentes couleurs disponibles :

- Rouge
- Vert
- Bleu
- Jaune
- Bleu

Une structure, composée de trois attributs, permettant de définir une couleur RGB :

- Valeur de rouge
- Valeur de vert
- Valeur de bleu

Le constructeur effectue les tâches suivantes :

- Récupère les pattes passées en paramètre et les assigne aux attributs dédiés
- Définit les pattes de la LED et du buzzer en tant que sortie
- Initialise le tableau de structures de couleurs et définit les couleurs citées dans l'énumération

5.10.2 NOTIFICATION

La méthode permettant de créer une notification nécessitera 3 arguments lors de son appel :

- Une couleur
- Le nombre d'occurrence(s)
- L'état du buzzer

Fonctionnement :

- Itère n fois, en fonction du nombre d'occurrence(s)
 - Ecrit la couleur à l'aide de la méthode « `analogWrite(pin, value)` »
 - Attend (durée de la notification)
 - Eteint la LED
 - Si le buzzer est activé
 - Allume le buzzer (durée de la notification)

5.11 CLASSE C++ « MEASUREMENTS »

La classe « Measurements » permettra de stocker les N dernières mesures pour les valeurs minimales, maximales et moyennes de température et d'humidité. Pour stocker une mesure, nous utiliserons une structure, contenant deux attributs de type « `float` », afin garder une trace de la température et de l'humidité.

Le nombre de mesures (N) pouvant être stockées sur l'Arduino sera défini par une macro, permettant de définir N en fonction de la mémoire programme à disposition (70% au total dans notre cas).

Les statistiques seront réalisées pour une période de mesures donnée, à l'aide de la méthode « new_measurement(measure, timestamp) », qui permettra de définir les valeurs minimales, maximales et moyennes sur la période.

5.11.1 CLASSE ET CONSTRUCTEUR

La classe comporte 9 attributs :

- Une structure de mesures, de taille N, pour stocker toutes les valeurs moyennes
- Une structure de mesures, de taille N, pour stocker toutes les valeurs maximales
- Une structure de mesures, de taille N, pour stocker toutes les valeurs minimales
- Le nombre de mesures pris en compte dans la période
- La somme des mesures de température pour la période donnée
- La somme des mesures d'humidité pour la période donnée
- La limite de la période donnée [ms]
- L'index du buffer de la mesure actuelle
- Le nombre de mesures présentes dans le buffer

Le constructeur de la classe effectue les tâches suivantes :

- Initialise le tableau de mesures pour les valeurs minimales avec la constante « FLT_MAX »
- Initialise le tableau de mesures pour les valeurs maximales avec la constante « FLT_MIN »
- Initialise le tableau de mesures pour les valeurs moyennes avec la valeur 0
- Défini la première intervalle de mesures avec la fonction « millis() », en y ajoutant la valeur de la constante de la durée de la période (1000x60x60 [ms] → [h]).

5.11.2 NOUVELLE MESURE

Cette fonction permet d'envoyer, à chaque prise de mesures par les capteurs, le résultat vers la classe « Measurements ». Les données sont ensuite traitées afin de définir les valeurs minimales, maximales et moyennes pour la période donnée.

Fonctionnement :

- Pour chaque mesure envoyée à la fonction, vérifie le timestamp de celle-ci
- Si la mesure se situe dans l'intervalle :
 - Vérifie si valeur minimale (Température ou humidité)
 - Vérifie si valeur maximale
 - Incrémente le nombre de mesures
 - Incrémente la somme des températures pour la période
 - Incrémente la somme des humidités pour la période
- Sinon :
 - Calcule la moyenne pour la température (somme / nombre de mesures)

- Calcule la moyenne pour l'humidité (somme / nombre de mesures)
- Incrémente l'index du buffer ((index de la mesure courante + 1) % N)
- Défini la nouvelle période de mesures (millis() + durée de l'intervalle)
- Initialise les variables utilisées pour la somme des données, ainsi que le nombre de mesures prises en compte dans l'intervalle.

5.11.3 RECUPERER LE NOMBRE DE MESURES

Cette fonction permet de récupérer le nombre de mesures stockées dans les différents buffers (min / max / avg). Si le nombre de mesures est supérieur à la taille du buffer (N), elle retournera la valeur de N, sinon elle retournera le nombre de mesures réellement présentes dans le buffer.

5.11.4 RECUPERER L'INDEX DE LA DERNIERE MESURE

Cette fonction permet de retourner l'index du buffer où la dernière mesure a été stockée, après le calcul des moyennes de température et d'humidité.

5.11.5 RECUPERER UNE MESURE

Cette fonction permet de retourner une mesure, en fonction de l'index passé en paramètre.

La fonction nécessite deux paramètres :

- L'index de la mesure à retourner
- La référence de la mesure où devra être stockée les données de la mesure ciblée par l'index.

6 REALISATION

6.1 BLINK

Afin de faire clignoter une led, nous devons tout d'abord définir le numéro du pin sur lequel elle est connectée.

```
uint8_t LED_PIN = 13;
```

Pour nous définissons le pin en tant que sortie, il nous permet ainsi d'y faire « sortir » du courant. Dans le cas d'un bouton, par exemple, on l'aurait défini en tant qu'entrée (INPUT).

```
pinMode(LED_PIN, OUTPUT); // sets the pin as output
```

Nous pouvons dès à présent utiliser la led. Pour cela, nous pouvons utiliser la méthode « digitalWrite() », qui permet d'allumer une led (niveau logique haut) ou d'éteindre une led (niveau logique bas). Dans le cas d'un « HIGH », la tension de sortie du pin sera donc de 5 V, ce qui permet d'alimenter leds, capteurs, etc.

```
digitalWrite(LED_PIN, HIGH); // switches on the led
delay(1000); // waits for 1000ms -> 1s
digitalWrite(LED_PIN, LOW); // switches off the led
delay(1000);
```

Quant à la fonction « delay() », elle permet de mettre en pause le programme pendant un temps t [ms] donné.

6.2 LOGGING

6.2.1 LOGGING.CPP

Les méthodes « PrintHeader() » et PrintSerial » sont appelées par les différentes macros de logging et permettent d'afficher convenablement les messages et arguments passés à celles-ci.

La méthode « PrintHeader() » permet d'afficher le temps d'exécution jusqu'à l'appel de l'une des macros de logging. Nous utilisons la méthode « millis() » pour récupérer le temps d'exécution [ms], duquel nous pouvons en tirer les heures / minutes / secondes à l'aide de divisions et de modulus.

```
void PrintHeader(const char* szHeaderType) {
    // returns the number of ms since the Arduino began running
    unsigned long time = millis();
    Serial.print(szHeaderType);
    Serial.print("\t at time : ");
    Serial.print(time / 3600000); // hours
    time = time % 3600000;
    Serial.print("h ");
    Serial.print(time / 60000); // minutes
    time = time % 60000;
    Serial.print("m ");
    Serial.print((time / 1000) % 60); // seconds
    Serial.print("s : \t");
}
```

La méthode « PrintSerial() » permet simplement d'afficher deux paramètres sur la même ligne, ce qui n'est pas possible avec un seul « Serial.print() ».

```
void PrintSerial(const char* format, int arg1) {
    Serial.print(format);
    Serial.println(arg1);
}
```

6.2.2 LOGGING.H

Le fichier « Logging.h » contient les différentes macros de logging. Nous débutons par déclarer la constante de préprocesseur « LOGGING_H », importer les librairies nécessaires au bon fonctionnement du programme, à « récupérer » la variable « g_logging_level », instanciée dans le fichier « Logging.cpp » et à déclarer une constante « TR_LOGLEVEL » à destination du préprocesseur, afin de définir le niveau de logging et donc, le niveau de compilation des différentes macros ci-dessous. Par exemple, en assignant la valeur 0 à « TR_LOGLEVEL », nous obtiendrons le programme le plus léger ; cependant, nous disposerons d'aucune information de logging dans la console.

```
#pragma once
#ifndef LOGGING_H
#define LOGGING_H

#include <stdio.h>
#include "Arduino.h"

extern uint8_t g_logging_Level;

//logging macros
#ifndef TR_LOGLEVEL
#define TR_LOGLEVEL 3
#endif
```

On « importe » ensuite les deux méthodes écrites dans le fichier « Logging.cpp », afin de pouvoir les appeler depuis les macros.

```
void PrintHeader(const char* szHeaderType);
void PrintSerial(const char* format, int arg1);
```

Ci-dessous, les deux macros permettant d'afficher des logs à titre informatif, qui nécessitent un (« TraceInfo() ») ou deux arguments (« TraceInfoFormat »). Le « résultat » de ces deux macros sera visible dans la console uniquement si la variable destinée au préprocesseur « TR_LOGLEVEL » est plus grande ou égale à 2 et que le programmeur ait décidé d'afficher ces logs (« g_logging_level » ≥ 2).

```
#if(TR_LOGLEVEL >= 2)
#define TraceInfo(format) if (g_logging_Level >= 2) {PrintHeader("INFO"); Serial.println(format);}
#define TraceInfoFormat(format, arg1) if (g_logging_Level >= 2) { PrintHeader("INFO"); PrintSerial(format, arg1);}
#else
#define TraceInfo(format)
#define TraceInfoFormat(format, arg1)
#endif
```

Le « # » permet de définir des instructions pour le préprocesseur.

6.3 MEMOIRE LIBRE

Comme précédemment expliqué dans la partie conception, la mémoire libre peut être déterminée en soustrayant l'adresse d'une variable fraîchement instanciée par la première adresse disponible (« `__heap_start` »).

```
int displayFreeRam() {
    extern int __heap_start;
    uint8_t free_memory;
    return ((int)&free_memory) - ((int)&__heap_start);
}
```

(Corrections TP01) En utilisant le « `heap_start` », nous ne pouvons que traiter le cas où de la mémoire dynamique est allouée. Afin de palier à ce manqueument, nous pouvons utiliser la « `_brkval` ». Exemple¹⁴ :

```
int freeMemory() {
    int free_memory;
    if ((int)_brkval == 0) {
        free_memory = ((int)&free_memory) - ((int)&__heap_start);
    } else {
        free_memory = ((int)&free_memory) - ((int)_brkval);
        free_memory += freeListSize();
    }
    return free_memory;
}
```

6.4 UTILISATION DES CAPTEURS

6.4.1 DHT11

Afin d'utiliser le capteur, il nous faut tout d'abord importer la librairie, réalisée par Adafruit , définir la broche utilisée par le capteur et créer une instance d'un capteur DHT11 :

```
#include "DHT.h"
#define DHTPIN 8
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
```

(Compléments) constructeur de la classe DHT :

```
DHT::DHT(uint8_t pin, uint8_t type, uint8_t count) {
    _pin = pin;
    _type = type;
    _count = count;
    firstreading = true;
}
```

On peut ensuite passer à l'initialisation du capteur, dans la boucle « `setup()` », à l'aide de la fonction « `begin()` » :

```
dht.begin();
```

(Compléments) méthode « `begin()` » :

```
void DHT::begin(void) {
    // set up the pins!
    pinMode(_pin, INPUT);
    digitalWrite(_pin, HIGH);
    _lastreadtime = 0;
}
```

¹⁴ <http://playground.arduino.cc/code/AvailableMemory>

Nous pouvons dès lors passer à la prise de mesures, à l'aide des méthodes « dht.readTemperature() » et « dht.readHumidity() » :

```
h = dht.readHumidity();
t = dht.readTemperature();
```

(Compléments) le code des méthodes « readTemperature() » et « readHumidity() » est disponible dans le fichier « DHT.cpp », livré avec la librairie.

6.4.2 PHOTORESISTANCE

La photorésistance étant un capteur analogique, il très aisément de recueillir la mesure effectuée par celle-ci à l'aide la méthode « analogRead(A0) » :

```
l = analogRead(A0);
```

Afin de « calibrer » les valeurs lues par l'Arduino, il peut être intéressant de mesurer des valeurs référence. Cette phase se déroulera sur les 5 premières secondes d'exécution du programme, où le capteur sera continuellement lu et permettra, si la valeur est inférieure ou supérieure aux bornes en place (Low level : 0 / High level : 1023), de les modifier.

```
// for calibration ...
int lowLight = 0;
int highlight = 1023;

void setup() {
    while (millis() < 5000) {
        l = analogRead(A0);
        if (l > highlight)
            highlight = l;
        if (l < lowLight)
            lowLight = l;
    }
}
```

Nous pouvons ensuite étalonner les valeurs sur une échelle plus réduite, en prenant en considération les deux variables précédemment calibrées, à l'aide de la fonction « map() ».

```
l = map(l, lowLight, highlight, 0, 100);
```

6.5 BLUETOOTH LE – NRF8001

Afin d'utiliser la puce nRF8001, nous débutons par importer les différentes librairies nécessaires au bon fonctionnement de la puce radio et nous incluons les fichiers d'entêtes correspondants. Nous incluons également le fichier « BLEconnect_TxPower.h » ou le fichier « BLEboradcast_TxPower.h », en fonction du mode de fonctionnement désiré (mode connecté / broadcast).

```
#include <SPI.h>
#include <aci_cmds.h>
#include <aci_evts.h>
#include <hal_aci_t1.h>
#include <lib_aci.h>
#include <EEPROM.h>

#include "NRF8001Device.h"
#include "_BLE_connect_TxPower.h"
//#include "_BLE_broadcast_TxPower.h"
```

Nous définissons ensuite des macros, sous forme de paramètres, pour linstanciation dun objet « nRF8001Device » et les broches où est connectée la puce radio.

```
static const hal_aci_data_t setup_msgs[NB_SETUP_MESSAGES] PROGMEM = SETUP_MESSAGES_CONTENT;
// for storing the service pipe data created in nRFgo Studio
#ifndef SERVICES_PIPE_TYPE_MAPPING_CONTENT
    static services_pipe_type_mapping_t services_pipe_type_mapping[NUMBER_OF_PIPES] =
SERVICES_PIPE_TYPE_MAPPING_CONTENT;
#else
    #define NUMBER_OF_PIPES 0
    static services_pipe_type_mapping_t* services_pipe_type_mapping = NULL;
#endif

#define REQ 10
#define RDY 2
#define RST 9
```

Nous instancions ensuite un objet de la classe « nRF8001Device », qui nous permettra dutiliser la puce Bluetooth.

```
// creates a instance of nRF8001Device
nRF8001Device nrf(
    setup_msgs,
    NB_SETUP_MESSAGES,
    services_pipe_type_mapping,
    NUMBER_OF_PIPES,
    nRF8001Device::CONNECT,
    REQ,
    RDY,
    RST);

// ACI event status on the nRF8001
aci_evt_opcode_t lastStatus = ACI_EVT_DISCONNECTED;
```

Nous initialisons ensuite lobjet créé précédemment dans la fonction « setup() », à l'aide des méthodes « setDeviceName() » et « begin() ».

```
// sets the device name
nrf.setDeviceName("IoT_Samuel");
// starts the nRF8001
bool nrfIsStarted = nrf.begin(1, 0x0100, 0);
if (nrfIsStarted == false)
    TraceError(F("nRF8001 device can't be started"));
```

Nous pouvons ensuite utiliser les différentes méthodes relatives à la puce radio dans la boucle principale (« loop() ») du sketch. Nous récupérons ensuite létat dans lequel se trouve lACI et nous laffichons sur la console.

```
// gets all aci events at regular intervals
nrf.pollACI();

// gets the device state
aci_evt_opcode_t status = nrf.getState();

if (status != lastStatus) {
    if (status == ACI_EVT_DEVICE_STARTED) {
        TraceInfo(F("Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        TraceInfo(F("Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        TraceInfo(F("Disconnected or advertising timed out"));
    }
    // updates the last status to this one
    lastStatus = status;
}
```

Si une connexion est active, nous affichons sur la console la mesure actuelle (température, humidité et luminosité). Le code nécessaire à la prise de mesures est défini dans des macros, ce qui permet dactiver ou non les mesures et ainsi, d'économiser de la mémoire.

```

if (status == ACI_EVT_CONNECTED) {
    #if (MEASURING)
        // reads the sensors
        h = dht.readHumidity();
        t = dht.readTemperature();
        l = analogRead(A0);

        // checks if any reads failed and exit early (to try again).
        if (isnan(h) || isnan(t) || isnan(l)) {
            TraceInfo(F("Failed to read from DHT sensor or from the photocell!"));
            return;
        }
        // displays the measures
        Serial.print(F("Humidity: "));
        Serial.print(h);
        Serial.print(F(" % Temperature: "));
        Serial.print(t);
        Serial.print(F(" *C Light level: "));
        Serial.print(l);
        Serial.println(F(" [0 - 1023]"));
        delay(5000);
    #endif
}

```

Les définitions nécessaires à la prise de mesures, déclarées en amont des méthodes « `setup()` » et « `loop()` ».

```

// 1/0 : On/Off
#ifndef MEASURING
#define MEASURING 1
#endif

#if (MEASURING)
    #include "DHT.h"
    #define DHTPIN 8
    #define DHTTYPE DHT11
    DHT dht(DHTPIN, DHTTYPE);
    // variables for the measures...
    float h = 0; // humidity
    float t = 0; // temperature
    float l = 0; // light
#endif

```

Et l'initialisation du capteur DHT11 se fait à l'intérieur de la boucle « `setup()` ».

```

#if (MEASURING)
    dht.begin(); // initialization the DHT11
#endif

```

6.6 CLASSE C++ SPECIFIQUE A LA STATION METEO

Le code source étant disponible sur GIT (cf. annexes), seuls les concepts fondamentaux seront détaillés ci-dessous.

Définition des structures de données nécessaires à l'envoi de la température et de l'humidité (.h) :

```

// Temperature measurement Flags
// Documentation:
// https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.temperature_measurement.xml
// Flags: 0: Celsius, 0: Time Stamp not present, 0:Temperature Type not present, 3-7: reserved
#define TEMPERATURE_MEASUREMENT_FLAGS 0b00000000

// Temperature measurement structure (5 bytes)
typedef struct temperature_measure {
    uint8_t flags;
    uint8_t measurement[4];
}temperature_measure;

// Humidity measurement structure (2 bytes)
// Documentation:
// https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.humidity.xml
typedef struct humidity_measure {
    uint8_t measurement[2];
}humidity_measure;

```

6.6.1 CLASSE ET CONSTRUCTEUR

Pour créer une classe dérivée, il suffit d'utiliser l'instruction « class nRF8001MeteoStation (nouvelle classe) :public nRF8001Device (classe mère) ». Dès lors, la nouvelle classe hérite de tous les attributs et fonctions de la classe mère (sauf si private).

La méthode « onACIEvent(...) », déclarée dans le fichier « nRF8001Device.h », est une fonction dite virtuelle ; elle nécessite d'être définie dans la classe dérivée et est appelée dans la méthode « pollACI() » (classe « nRF8001Device », événements : « ACL_EVT_DISCONNECTED » et « ACL_EVT_DATA_ACK »).

```
// Derived class of nRF8001Device
// This class specifies the needed functions & attributs used for our Wheater Station
class nRF8001MeteoStation:public nRF8001Device {
public:
    // Constructor
    nRF8001MeteoStation();

    /**
     * Function is called on defined ACI events (cf. nRF8001Device.cpp)
     * @param aci_evt_t* Pointer to the aci_data event structure
     */
    virtual void onACIEvent(aci_evt_t* p_event);

private:
    // variables used to store the last temperature & humidity measurements
    float m_last_temp;
    float m_last_hum;
    ...
};
```

Le constructeur ne nécessite aucun paramètre. Dans celui-ci, nous appelons le constructeur de la classe mère et nous initialisons les membres nouvellement créés pour cette classe.

- Dernières températures envoyées → FLT_MAX
- Allocation de la mémoire nécessaire aux structures
- Acquittements en suspens → FALSE

```
nRF8001MeteoStation::nRF8001MeteoStation():
    nRF8001Device (setup_msgs, NB_SETUP_MESSAGES, services_pipe_type_mapping, NUMBER_OF_PIPES,
nRF8001Device::CONNECT, REQ, RDY, RST) {
    // initializes the last temperature & humidity to the biggest FLOAT value
    m_last_temp = FLT_MAX;
    m_last_hum = FLT_MAX;

    // initializes the memory space taken by the measurement's structures
    memset(&m_temp_measure, 0, sizeof(m_temp_measure));
    memset(&m_hum_measure, 0, sizeof(m_hum_measure));

    // sets the ACKs to FALSE; no ACKs are pending
    m_ack_temperature_measure_pending = false;
    m_ack_humidity_measure_pending = false;
}
```

6.6.2 ENVOI DES DONNEES

L'envoi des données nécessite que plusieurs conditions soient remplies :

- Donnée précédemment envoyée soit acquittée
- Donnée à envoyer différente de la donnée précédemment envoyée
- Pipe dédié à la donnée soit disponible

Ces conditions sont vérifiées à l'aide des deux « if », au début de la fonction ; si elles sont remplies, la donnée est ensuite assignée à la structure puis envoyée à l'aide la fonction mise à disposition par la librairie BLE « lib_aci_send_data(...) ».

```

bool nRF8001MeteoStation::send_temperature(float p_temperature) {
    // if the last sending was acquitted and the temperature to send is different
    // from the earlier temperature, it checks if the pipe is available
    if ((m_ack_temperature_measure_pending == false) && (p_temperature != m_last_temp)) {
        if (lib_aci_is_pipe_available(&aci_state, PIPE_METEO_STATION_TEMPERATURE_MEASUREMENT_TX_ACK)) {
            // sets the structure with the corresponding flags (cf. .h) and the temperature with the given
            parameter
            m_temp_measure.flags = TEMPERATURE_MEASUREMENT_FLAGS;
            m_temp_measure.measurement[0] = p_temperature;
            m_temp_measure.measurement[1] = 0;
            m_temp_measure.measurement[2] = 0;
            m_temp_measure.measurement[3] = 0;

            // sends the temperature structure (Pipe, structure address, size)
            m_ack_temperature_measure_pending =
                lib_aci_send_data(PIPE_METEO_STATION_TEMPERATURE_MEASUREMENT_TX_ACK, (uint8_t *)&m_temp_measure, 5);
            // if the value was sent correctly, it stores the temperature
            if (m_ack_temperature_measure_pending)
                m_last_temp = p_temperature;
            return m_ack_temperature_measure_pending;
        }
    }
    return false;
}

```

Le principe est identique pour l'envoi d'une humidité.

6.6.3 RECEPTION DES ACQUITTEMENTS

Les acquittements peuvent être ensuite aperçus à l'aide la fonction « onACIEvent(...) ». Pour rappel, cette méthode est appelée à l'intérieur de la méthode « pollACI() » et permet d'identifier le numéro du Pipe sur lequel l'acquittement a été transmis.

```

void nRF8001MeteoStation::onACIEvent(aci_evt_t* p_event) {
    switch(p_event->evt_opcode) {
        // if an ACI_EVT_DATA_ACK occurs, it searches the pipe, which has return an ACK
        case ACI_EVT_DATA_ACK:
            if (p_event->params.data_ack.pipe_number == PIPE_METEO_STATION_TEMPERATURE_MEASUREMENT_RX_ACK) {
                // sets to False; a new sending could take place in case of a new temperature (!= m_last_temp)
                m_ack_temperature_measure_pending = false;
                TraceInfo(F("ACK for temperature measurement received\n"));
            } else if (p_event->params.data_ack.pipe_number == PIPE_METEO_STATION_HUMIDITY_MEASUREMENT_RX_ACK) {
                m_ack_humidity_measure_pending = false;
                TraceInfo(F("ACK for humidity measurement received\n"));
            }
            break;
        // if a paired device is disconnected, it sets the stored values to the biggest FLOAT value
        // and the ACKs to false. It avoids problems if the device want to connect again
        case ACI_EVT_DISCONNECTED:
            m_last_temp = FLT_MAX;
            m_last_hum = FLT_MAX;
            m_ack_temperature_measure_pending = false;
            m_ack_humidity_measure_pending = false;
            break;
    }
}

```

6.7 CLASSE C++ « BUTTON »

Le code source étant disponible sur GIT (cf. annexes), seuls les concepts fondamentaux seront détaillés ci-dessous.

6.7.1 CLASSE ET CONSTRUCTEUR

Les attributs nécessaires au bon fonctionnement de la classe « Button » sont déclarés dans le fichier d'entêtes « Button.h » et sont au nombre de 8. Pour plus d'informations à leur propos, le point 5.9 du présent document décrit leur utilité.

La définition des attributs, en tant que membres privés :

```
private:
    // variables used to store the pin of the button & the timeout
    uint8_t m_button_pin;
    uint16_t m_click_ticks;

    // callback functions (single-click / double-click)
    callbackFunction m_click_fnct;
    callbackFunction m_double_click_fnct;

    // variables used to hold information across the upcoming tick calls
    unsigned long m_start_time;
    unsigned long m_now;

    uint8_t m_state;
    uint8_t m_button_level;    // LOW / HIGH
```

A la création d'une instance d'un objet de la classe « Button », le constructeur va assigner la patte passée en paramètre à l'attribut dédié et initialiser les autres attributs.

```
button::button(uint8_t p_button_pin):
    m_button_pin(p_button_pin),
    m_click_ticks(800),
    m_state(0),
    m_button_level(LOW) {
    // sets the pin as INPUT
    pinMode(m_button_pin, INPUT);
}
```

6.7.2 DISTINCTION DU TYPE DE PRESSION

```

void button::tick() {
    // detects the input level & gets the current time
    m_button_level = digitalRead(m_button_pin);
    m_now = millis();

    if (m_state == 0) {
        if (m_button_level == HIGH) {
            m_state = 1;
            m_start_time = m_now; // remembers starting time
        }
    } else if (m_state == 1) {
        if (m_button_level == LOW) {
            m_state = 2;
        }
    } else if (m_state == 2) {
        if (m_now > m_start_time + m_click_ticks) {
            // this was only a single-click
            if (m_click_fnct) m_click_fnct();
            m_state = 0; // resets state
        } else if (m_button_level == HIGH) {
            m_state = 3;
        }
    } else if (m_state == 3) {
        if (m_button_level == LOW) {
            // this was a 2 click sequence.
            if (m_double_click_fnct) m_double_click_fnct();
            m_state = 0; // resets state
        }
    }
}

```

6.8 CLASSE C++ « NOTIFICATION »

Le code source étant disponible sur GIT (cf. annexes), seuls les concepts fondamentaux seront détaillés ci-dessous.

6.8.1 CLASSE ET CONSTRUCTEUR

Les attributs nécessaires au bon fonctionnement de la classe « Notification » sont déclarés dans le fichier d'entêtes « Notification.h » et sont au nombre de 8. Pour plus d'informations à leur propos, le point 5.10 du présent document décrit leur utilité.

La définition de l'énumération comportant les couleurs à disposition, ainsi que de la structure permettant de stocker une couleur sous la forme RGB (3 bytes) :

```
// Enumeration of the different available colors
enum Colors {
    RED,
    GREEN,
    BLUE,
    YELLOW,
    WHITE,
    NB_OF_COLOURS
};

// Structure to store the RGB components of a color
typedef struct Color {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
}Color;
```

La définition des attributs, en tant que membres privés :

```
private:
    // the pins where the LED & the buzzer are connected to
    uint8_t m_red_led_pin;
    uint8_t m_green_led_pin;
    uint8_t m_blue_led_pin;
    uint8_t m_buzzer_pin;

    // variables used to store the duration & interval of a notification
    uint16_t m_duration;
    uint16_t m_interval;

    // array of available colors
    Color m_colors[NB_OF_COLOURS] PROGMEM;

    // variable used to store the buzzer state (enabled / disabled)
    bool m_buzzer_enabled;
```

Le constructeur, qui nécessite 4 paramètres, va :

- Assigner le numéro des pattes passé en paramètre, aux différents attributs dédiés
- Initialise la mémoire utilisée par la structure des couleurs
- Définir les pattes en tant qu'entrée
- Définir les couleurs, avec leur valeur RGB

```

notification::notification(uint8_t p_red_led_pin, uint8_t p_green_led_pin, uint8_t p_blue_led_pin, uint8_t
p_buzzer_pin):
    m_red_led_pin(p_red_led_pin),
    m_green_led_pin(p_green_led_pin),
    m_blue_led_pin(p_blue_led_pin),
    m_buzzer_pin(p_buzzer_pin),
    m_duration(DURATION),
    m_interval(INTERVAL),
    m_buzzer_enabled(false) {
    // initializes the memory space taken by the colors' array
    memset(&m_colors, 0, sizeof(m_colors));

    // sets the pins as OUTPUT
    pinMode(m_red_led_pin, OUTPUT);
    pinMode(m_green_led_pin, OUTPUT);
    pinMode(m_blue_led_pin, OUTPUT);
    pinMode(m_buzzer_pin, OUTPUT);

    // defines the colors
    m_colors[RED].red = m_colors[GREEN].green = m_colors[BLUE].blue = m_colors[YELLOW].red =
m_colors[YELLOW].green = m_colors[WHITE].red = m_colors[WHITE].green = m_colors[WHITE].blue = 255;
    m_colors[GREEN].red = m_colors[BLUE].green = 128;
    m_colors[RED].green = m_colors[RED].blue = m_colors[GREEN].blue = m_colors[BLUE].red =
m_colors[YELLOW].blue = 0;
}

```

6.8.2 NOTIFICATION

La méthode « notify » nécessite 3 paramètres : une couleur, un nombre d'occurrence(s) et un booléen, permettant de forcer l'utilisation du buzzer, employé normalement uniquement pour des notifications critiques (température inférieure ou supérieure au thermostat).

La fonction « allume » la LED en précisant les valeurs pour les canaux RGB, puis attend un certain temps (intervalle donné) puis éteint la LED.

```

void notification::notify(Colors p_color, uint8_t p_occurrences, bool p_buzzer_enabled) {
    for (uint8_t i = 0; i < p_occurrences; i++) {
        // switches on the LED
        analogWrite(m_red_led_pin, m_colors[p_color].red);
        analogWrite(m_green_led_pin, m_colors[p_color].green);
        analogWrite(m_blue_led_pin, m_colors[p_color].blue);

        delay(m_interval);

        // switches off the LED
        digitalWrite(m_red_led_pin, LOW);
        digitalWrite(m_green_led_pin, LOW);
        digitalWrite(m_blue_led_pin, LOW);

        if (p_buzzer_enabled || m_buzzer_enabled)
            tone(m_buzzer_pin, 4978, m_duration);
    }
}

```

6.9 CLASSE C++ « MEASUREMENTS »

Le code source étant disponible sur GIT (cf. annexes), seuls les concepts fondamentaux seront détaillés ci-dessous.

La structure de données est composée de deux types « float », pour un total de 8 bytes.

```
// Temperature & humidity measurements structure
// float: 4 bytes -> 8 bytes
typedef struct measure_value {
    float temperature;
    float humidity;
}measure_value;
```

Pour définir N, il est nécessaire de compiler le code avec la valeur 1, afin de déterminer la mémoire utilisée par les différentes variables au sein du programme. Nous calculons ensuite le nombre maximal de bytes à disposition (70% de la mémoire programme ; 2048 / 10 x 7). Nous soustrayons ensuite cette valeur avec la valeur trouvée auparavant, -24 (3 structures de mesures ; N = 1).

Nous trouvons donc la valeur de 10 pour N :

```
// 2048 / 10 x 7 = 1433.6 (70%)
// 1149 - 24 = space used without the N structures
// 1433.6 - 1125 = 260 / 24 = 10
#define N 10
```

6.9.1 CLASSE ET CONSTRUCTEUR

La classe comporte 9 attributs, déclarés en privé:

```
private:
    // structures used to store the historical measurements
    measure_value m_mea_avg[N];
    measure_value m_mea_min[N];
    measure_value m_mea_max[N];

    // variables used to sum and compute the avg of temperature & humidity
    uint16_t m_nb_of_mea_stored;      // number of measurements used for the avg
    float m_temp_sum;
    float m_hum_sum;

    unsigned long m_end_measure;     // end of the measurement [ms]
    uint16_t m_current_measure;      // index in the circular buffer
    uint8_t m_nb_of_measurements;    // total number of measurements
```

Le constructeur de la classe effectue les tâches suivantes :

- Initialise le tableau de mesures pour les valeurs minimales avec la constante « FLT_MAX »
- Initialise le tableau de mesures pour les valeurs maximales avec la constante « FLT_MIN »
- Initialise le tableau de mesures pour les valeurs moyennes avec la valeur 0
- Défini la première intervalle de mesures avec la fonction « millis() », en y ajoutant la valeur de la constante de la durée de la période (1000x60x60 [ms] → [h]).

```

measurements::measurements():
    m_nb_of_mea_stored(0),
    m_current_measure(0),
    m_temp_sum(0),
    m_hum_sum(0),
    m_nb_of_measurements(0) {
        // initialization of the buffers
        for (uint16_t i = 0; i < N; i++) {
            m_mea_avg[i].temperature = 0;
            m_mea_avg[i].humidity = 0;

            m_mea_min[i].temperature = FLT_MAX;
            m_mea_min[i].humidity = FLT_MAX;

            m_mea_max[i].temperature = FLT_MIN;
            m_mea_max[i].humidity = FLT_MIN;

            // defines the measurement period
            m_end_measure = millis() + MEASURE_INTERVAL;
        }
    }
}

```

6.9.2 NOUVELLE MESURE

La fonction « new_measurement(...) » permet d'envoyer, à chaque prise de mesures par les capteurs, le résultat vers la classe « Measurements ». Les données sont ensuite traitées afin de définir les valeurs minimales, maximales et moyennes pour la période donnée.

Elle nécessite en paramètre la référence vers une mesure de type « measure_value ». :

```

void measurements::new_measurement(measure_value& p_measure, unsigned long p_timestamp) {
    // if the new measurement is part of the current period, compute the min / max values
    if (p_timestamp <= m_end_measure) {
        if (p_measure.temperature < m_mea_min[m_current_measure].temperature)
            m_mea_min[m_current_measure].temperature = p_measure.temperature;
        if (p_measure.humidity < m_mea_min[m_current_measure].humidity)
            m_mea_min[m_current_measure].humidity = p_measure.humidity;

        if (p_measure.temperature > m_mea_max[m_current_measure].temperature)
            m_mea_max[m_current_measure].temperature = p_measure.temperature;
        if (p_measure.humidity > m_mea_max[m_current_measure].humidity)
            m_mea_max[m_current_measure].humidity = p_measure.humidity;

        m_nb_of_mea_stored++;
        m_temp_sum += p_measure.temperature;
        m_hum_sum += p_measure.humidity;
    } else {
        // computes the avg (measurements sum / number of measurements)
        m_mea_avg[m_current_measure].temperature = m_temp_sum / m_nb_of_mea_stored;
        m_mea_avg[m_current_measure].humidity = m_hum_sum / m_nb_of_mea_stored;

        TraceInfoFormat(F("Temperature:\tmin / max / avg : %u / %u / %u\n"),
        (uint8_t)m_mea_min[m_current_measure].temperature, (uint8_t)m_mea_max[m_current_measure].temperature,
        (uint8_t)m_mea_avg[m_current_measure].temperature);
        TraceInfoFormat(F("Humidity:\tmin / max / avg : %u / %u / %u\n"),
        (uint8_t)m_mea_min[m_current_measure].humidity, (uint8_t)m_mea_max[m_current_measure].humidity,
        (uint8_t)m_mea_avg[m_current_measure].humidity);

        m_current_measure = (m_current_measure + 1) % N; // circular buffer, +1 % size
        m_nb_of_measurements++;

        // initializes the current structure (in case of override, m_nb_of_measurements > N)
        m_mea_max[m_current_measure].temperature = FLT_MIN;
        m_mea_max[m_current_measure].humidity = FLT_MIN;
        m_mea_min[m_current_measure].temperature = FLT_MAX;
        m_mea_min[m_current_measure].humidity = FLT_MAX;

        m_end_measure = millis() + MEASURE_INTERVAL;
        // initializes the number of measurements used for the avg
        m_nb_of_mea_stored = 0;
        m_temp_sum = 0;
        m_hum_sum = 0;

        TraceInfoFormat(F("Number of measure: %u Last measurement index: %u\n"), get_nb_measurements(),
        get_last_measurement_index());
    }
}

```

6.9.3 RECUPERER LE NOMBRE DE MESURES

Cette fonction permet de récupérer le nombre de mesures stockées dans les différents buffers (min / max / avg). Si le nombre de mesures est supérieur à la taille du buffer (N), elle retournera la valeur de N, sinon elle retournera le nombre de mesures réellement présentes dans le buffer.

```
uint8_t measurements::get_nb_measurements(void) {
    if (_nb_of_measurements > (N - 1))
        return N;
    else
        return _nb_of_measurements;
}
```

6.9.4 RECUPERER L'INDEX DE LA DERNIERE MESURE

Cette fonction permet de retourner l'index du buffer où la dernière mesure a été stockée, après le calcul des moyennes de température et d'humidité.

```
uint8_t measurements::get_last_measurement_index(void) {
    if (_nb_of_measurements < N && _current_measure == 0)
        return 0;
    else if (_current_measure == 0)
        return N-1;
    return _current_measure - 1;
}
```

6.9.5 RECUPERER UNE MESURE

Cette fonction permet de retourner une mesure, en fonction de l'index passé en paramètre.

La fonction nécessite deux paramètres :

- L'index de la mesure à retourner
- La référence de la mesure où devra être stockée les données de la mesure ciblée par l'index.

```
void measurements::get_measurement(uint8_t p_index, measure_value& p_measure) {
    p_measure.temperature = _mea_avg[p_index].temperature;
    p_measure.humidity = _mea_avg[p_index].humidity;
}
```

7 TESTS ET VALIDATION

7.1 TP01

Afin d'attester le bon fonctionnement des macros de logging, nous pouvons incorporer des appels vers celles-ci à l'intérieur du code réalisé pour les étapes de ce présent TP. Nous utilisons donc les méthodes « `TraceErrorFormat()` », « `TraceInfoFormat()` » et `TraceDebugFormat()` » et les constantes « `TR_LOGLEVEL` » et « `g_logging_level` » afin de faire varier la taille du programme compilé et l'affichage en sortie sur la console.

En définissant un « `TR_LOGLEVEL` » à 0, nous obtenons un code compilé d'une taille de 184 bytes, ce qu'il illustre le fait qu'aucune des macros n'a été compilée et donc, nous obtenons aucun message sur la console. Mais en définissant des niveaux de logging plus élevés, nous pouvons constater, comme le montrent très bien les figures ci-dessous, que la taille des programmes augmente, ainsi que la quantité d'information affichée sur la console.

7.1.1 LOGGING DE NIVEAU 1

Niveau de logging à 1 : affichage sur la console que des messages d'erreur ; taille des données de 242 bytes.

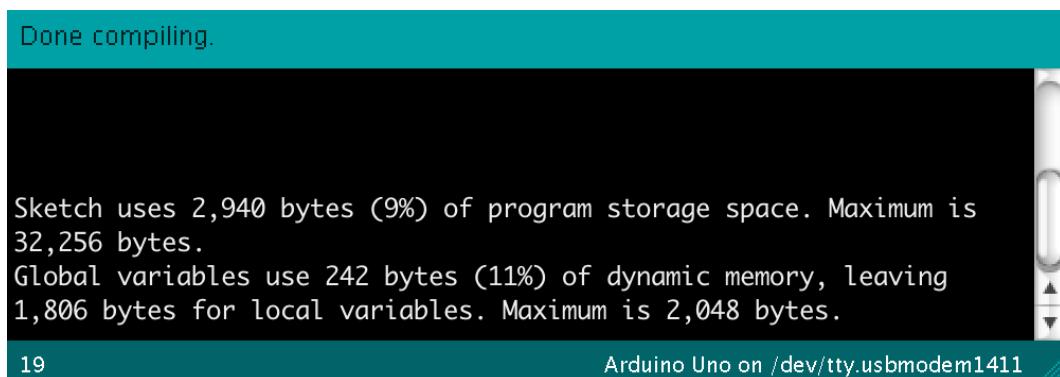


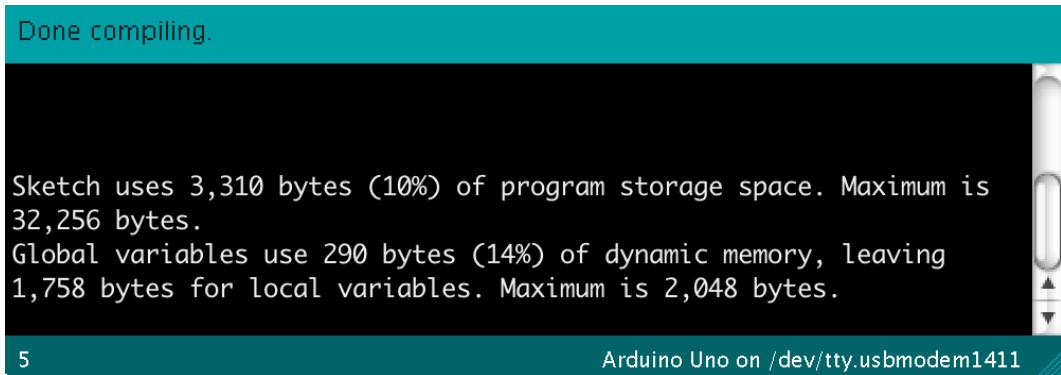
Figure 39: Compilation du programme avec un niveau de logging de 1

```
ERROR at time : 0h 0m 44s : Print an error message ...
ERROR at time : 0h 0m 1s : Print an error message ...
ERROR at time : 0h 0m 4s : Print an error message ...
ERROR at time : 0h 0m 6s : Print an error message ...
ERROR at time : 0h 0m 8s : Print an error message ...
ERROR at time : 0h 0m 10s : Print an error message ...
```

Figure 40: Sortie sur la console

7.1.2 LOGGING DE NIVEAU 2

Niveau de logging à 2 : affichage sur la console des messages d'erreur et d'information; taille des données de 290 bytes.



The screenshot shows the Arduino IDE's serial monitor window. At the top, a teal bar displays "Done compiling.". The main black area contains the following text:

```
Sketch uses 3,310 bytes (10%) of program storage space. Maximum is 32,256 bytes.
Global variables use 290 bytes (14%) of dynamic memory, leaving
1,758 bytes for local variables. Maximum is 2,048 bytes.
```

At the bottom, a teal footer bar shows the number "5" on the left and "Arduino Uno on /dev/tty.usbmodem1411" on the right.

Figure 41: Compilation du programme avec un niveau de logging de 2


The screenshot shows the Arduino IDE's serial monitor window titled "/dev/tty.usbmodem1411 (Arduino Uno)". The main black area contains the following log output:

```
INF 0s : Free memory : 1751
INFO at time : 0h 0m 0s : The led is switched on; #1
INFO at time : 0h 0m 0s : Free memory : 1751
INFO at time : 0h 0m 0s : The led is switched on; #1
ERROR at time : 0h 0m 2s : Print an error message ...
INFO at time : 0h 0m 2s : The led is switched on; #2
ERROR at time : 0h 0m 4s : Print an error message ...
```

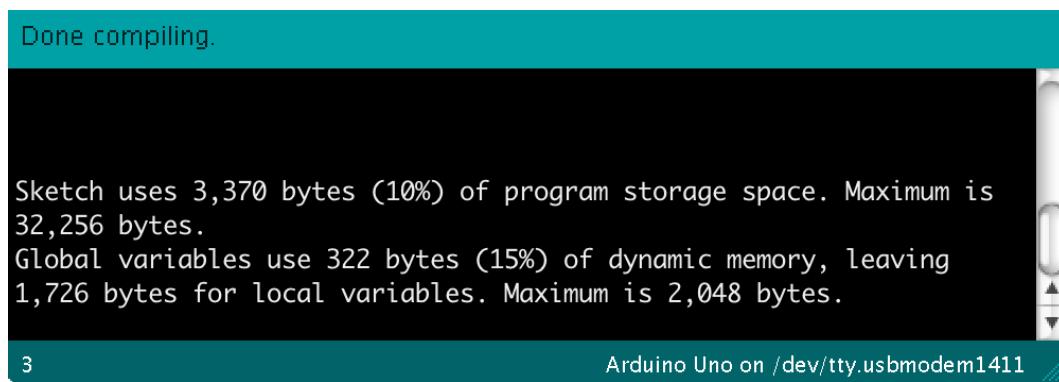
On the right side, there is a vertical scroll bar. A "Send" button is located in the top right corner of the window.

Figure 42: Sortie sur la console

7.1.3 LOGGING DE NIVEAU 3

Niveau de logging à 3 : affichage sur la console de tous les types de logging (ERROR, INFO & DEBUG) ; taille des données de 242 bytes.

Avec cette figure, nous pouvons aussi attester du bon fonctionnement de la méthode de calcul de la mémoire libre, qui retourne une valeur de 1719. Ce n'est, certes, pas identique à la valeur affichée dans la console, mais celle-ci en est très proche (+ 7 bytes).



The screenshot shows the Arduino IDE's serial monitor window. At the top, a teal bar displays "Done compiling.". The main black area contains the following text:

```
Sketch uses 3,370 bytes (10%) of program storage space. Maximum is 32,256 bytes.
Global variables use 322 bytes (15%) of dynamic memory, leaving
1,726 bytes for local variables. Maximum is 2,048 bytes.
```

At the bottom, a teal footer bar shows the number "3" on the left and "Arduino Uno on /dev/tty.usbmodem1411" on the right.

Figure 43: Compilation du programme avec un niveau de logging à 3

```

INFO at time : 0h 0m 0s : Free memory : 1719
INFO at time : 0h 0m 0s : The led is switched on; #1
DEBUG at time : 0h 0m 2s : Print a debug message ...
ERROR at time : 0h 0m 2s : Print an error message ...
INFO at time : 0h 0m 2s : The led is switched on; #2
DEBUG at time : 0h 0m 4s : Print a debug message ...
ERROR at time : 0h 0m 4s : Print an error message ...

```

Figure 44: Sortie sur la console

Synthèse des tests réalisés :

- ✓ Affichage de la valeur du compteur à chaque itération
- ✓ Affichage des logs de niveau 1
- ✓ Affichage des logs de niveaux 1-2
- ✓ Affichage des logs de niveaux 1-3
- ✓ Variation de la taille du programme compilé en fonction du niveau de logging
- ✓ Affichage de la quantité de mémoire libre au démarrage du programme

7.2 TP02

Afin de vérifier le bon fonctionnement du programme, consistant à prendre des mesures de température, d'humidité et de luminosité à des intervalles régulières, nous pouvons utiliser la console, dans laquelle on écrit les résultats issus du capteur DHT11 et de la photorésistance.

```

// displays the values
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" % Temperature: ");
Serial.print(t);
Serial.print(F(" *C Light level: "));
Serial.print(l);
Serial.println(F(" [0 - 100]"));

```

Ce code nous permet d'obtenir les messages suivants dans la console :

```

Humidity: 52.50 % Temperature: 20.80 *C Light level: 61 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 34 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 65 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 28 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 30 [0 - 100]

```

Figure 45: Sortie sur la console

L'étape suivante consiste à réaliser des mesures sur une plus longue période afin de démontrer des variations de température, d'humidité ou luminosité.

Conditions de la mesure :

- Durée : 60 minutes ; une mesure par minute
- Capteurs : DHT22 (température + humidité) et photorésistance
- Conditions : Fenêtre ouverte / fermée ; Store ouvert / fermé ; Lampe éteinte / allumée

#	Time	T	H	L
0	72	19.4	45.2	76
1	137	19.4	45.2	76
2	203	19.4	45.2	78
3	268	19.4	45.2	79
4	333	19.4	45.2	76
5	399	19.4	45.2	75
6	464	19.4	45.2	76
7	530	19.4	45.2	76
8	595	19.4	45.2	79
9	660	19.4	45.2	80
10	726	19.4	45.2	80
11	791	19.4	45.2	81
12	856	19.4	45.2	81
13	922	19.4	45.2	81
14	987	19.4	45.2	1
15	1052	19.4	45.2	1
16	1118	19.4	45.2	0
17	1183	19.4	45.2	1
18	1249	17.1	38.2	1

#	Time	T	H	L
19	1314	17.1	38.2	1
20	1379	17.1	38.2	1
21	1445	17.1	38.2	1
22	1510	17.1	38.2	1
23	1576	17.1	38.2	1
24	1641	17.1	38.2	1
25	1706	17.1	38.2	1
26	1772	17.1	38.2	9
27	1837	17.1	38.2	96
28	1903	17.1	38.2	95
29	1968	17.1	38.2	95
30	2034	17.1	38.2	95
31	2099	17.1	38.2	96
32	2165	17.1	38.2	93
33	2230	17.1	38.2	90
34	2296	17.1	38.2	94
35	2361	17.1	38.2	86
36	2427	14.6	45.7	86
37	2492	14.6	45.7	88

#	Time	T	H	L
38	2558	14.6	45.7	89
39	2623	14.6	45.7	86
40	2689	14.6	45.7	85
41	2773	18.6	81.2	94
42	2838	18.6	81.2	96
43	2904	18.6	81.2	96
44	2969	18.6	81.2	79
45	3069	19.9	44	69
46	3134	19.9	44	80
47	3200	19.9	44	81
48	3265	19.9	44	79
49	3331	19.9	44	78
50	3396	19.9	44	77
51	3462	19.9	44	76
52	3527	19.9	44	75
53	3593	19.9	44	73
54	3658	19.9	44	73
		19.3	44.8	57.8

Figure 46: Mesures effectuées sur une période de 60 minutes

Les mesures ont été récoltées à l'aide d'un petit script réalisé en Java, permettant de lire les données provenant du câble USB, d'en extirper les valeurs et de les enregistrer dans un fichier .csv.

L'utilisation du script, disponible dans le dossier « tp02/ ¹⁵ », s'effectue à l'aide d'un terminal :

```
java -jar ArduinoDataCollector.jar /dev/tty.usbmodem1411 3600
```

- /dev/tty.usbmodem1411: port où l'Arduino est connectée
- 3600: durée, en seconds, de la mesure

Nous avons ensuite tout loisir de créer un graphique à partir de ces données.

¹⁵ <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/blob/master/tp02/ArduinoDataCollector.jar>

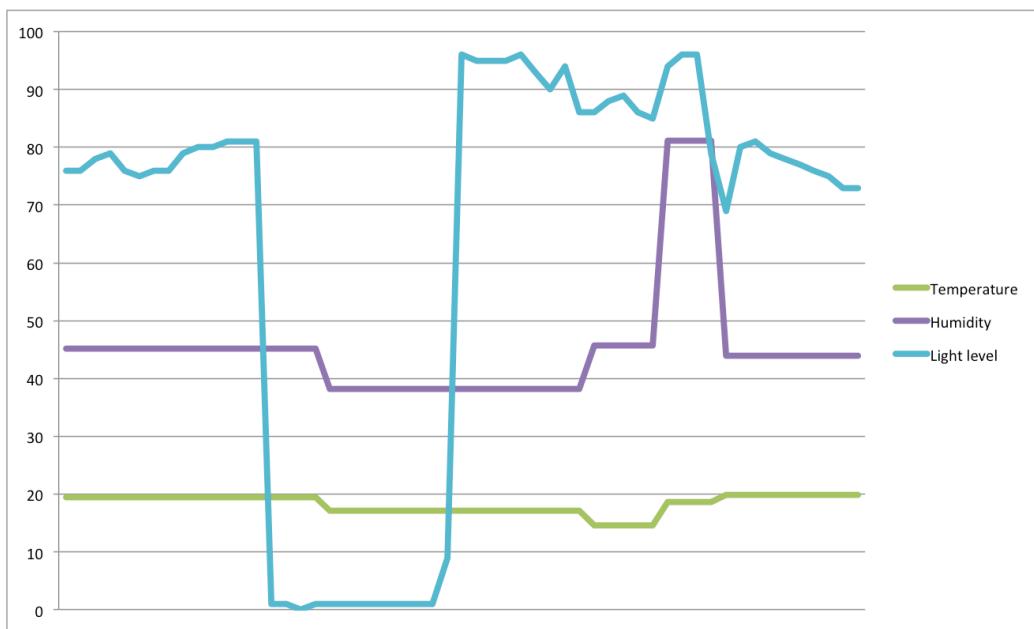


Figure 47: Graphique illustrant les variations des mesures effectuées précédemment

Remarque : Suite à l'implémentation de Watchdog, dans le but de limiter la consommation entre les mesures, le capteur DHT22 me semble moins réactif aux variations environnantes que précédemment. Cependant, la lecture de la photorésistance ne pose aucun problème.

Le programme est fonctionnel avec l'utilisation de Watchdog, cependant, n'ayant pas le matériel nécessaire, il m'a été impossible de mesurer les éventuelles et supposées, économies de courant.

Synthèse des tests réalisés :

- ✓ Affichage de la température, de l'humidité et de la luminosité sur la console
- ✓ Prise de mesures sur une période d'une heure, à l'aide d'un script réalisé en Java et traitement des données avec Excel
- ✓ Implémentation de Watchdog (optionnel)

7.3 TP03

Dans le but de vérifier le fonctionnement de la partie Bluetooth, nous débutons par utiliser l'entête « BLE_connect_TxPower.h » afin de rendre possible la connexion d'un périphérique à la puce radio.

En démarrant la console, nous pouvons constater que la puce est bien initialisée avec l'identifiant désiré (« IoT_Samuel ») et que celle-ci dispose d'une adresse physique.

```

oINFO at time 0 h 0 m 0 s 0 ms: Service type mapping setINFO at time 0 h 0 m 0 s 57 ms: Evt Device Started: Setup (nbr of credits available is 2)
INFO at time 0 h 0 m 0 s 231 ms: Evt Device Started: Standby (nbr of credits available is 2)
INFO at time 0 h 0 m 0 s 261 ms: Setting device name to IoT_Samuel
INFO at time 0 h 0 m 0 s 330 ms: Connect started
INFO at time 0 h 0 m 0 s 380 ms: Broadcasting started
INFO at time 0 h 0 m 0 s 437 ms: Advertising startedINFO at time 0 h 0 m 0 s 492 ms: Device address is e8:51:bf:97:8b: (type 2)
INFO at time 0 h 0 m 1 s 56 ms: Evt Disconnected
INFO at time 0 h 0 m 1 s 57 ms: Connect restarted

```

Figure 48: Initialisation de la puce nRF8001

Ensuite, à l'aide de l'application « nRF Master Control Panel » pour Android, nous pouvons lister et nous connecter à la station météo désirée, via le bouton « CONNECT».

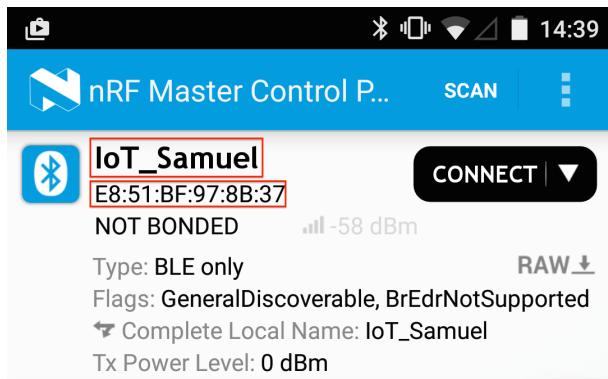


Figure 49: Liste des périphériques BLE en mode « connecté »

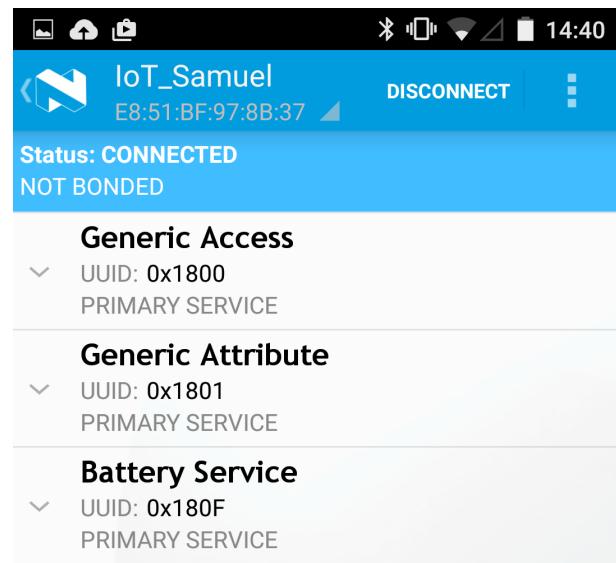


Figure 50: Caractéristiques fournies par la station météo

En s'y connectant, nous pouvons observer, dans l'application mobile, les différentes caractéristiques définies pour ce service, ainsi que le nom et l'adresse physique de la puce, identiques aux informations visibles sur la console. De plus, la connexion d'un périphérique conduit à la prise de mesures et à l'affichage de celles-ci sur la console.

```

INFO at time 0 h 0 m 15 s 50 ms: evt Disconnected
INFO at time 0 h 0 m 15 s 52 ms: Connect restarted
INFO at time 0 h 0 m 15 s 89 ms: Broadcasting restarted
INFO at time 0 h 0 m 16 s 51 ms: Evt Disconnected
INFO at time 0 h 0 m 16 s 52 ms: Connect restarted
INFO at time 0 h 0 m 16 s 89 ms: Broadcasting restarted
INFO at time 0 h 0 m 16 s 201 ms: Evt Connected
INFO at time 0 h 0 m 16 s 202 ms: Connected!Humidity: 42.00 % Temperature: 20.00 *C Light level: 921.00 [0 - 1023]
INFO at time 0 h 0 m 21 s 521 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x00000003
Humidity: 41.00 % Temperature: 18.00 *C Light level: 921.00 [0 - 1023]
Humidity: 44.00 % Temperature: 23.00 *C Light level: 923.00 [0 - 1023]
Humidity: 42.00 % Temperature: 20.00 *C Light level: 916.00 [0 - 1023]
Humidity: 41.00 % Temperature: 18.00 *C Light level: 923.00 [0 - 1023]

```

Autoscroll No line ending 9600 baud

Figure 51: Affichage des mesures lorsque qu'un périphérique est connecté au nRF8001

En incluant maintenant le fichier d'entêtes « BLE_broadcast_TxPower.h », nous pouvons constater la disparition du bouton « CONNECT ». C'est tout à fait normal, car nous nous trouvons maintenant en mode « broadcast ». Ce mode permet uniquement d'émettre des paquets et n'accepte pas de connexions.

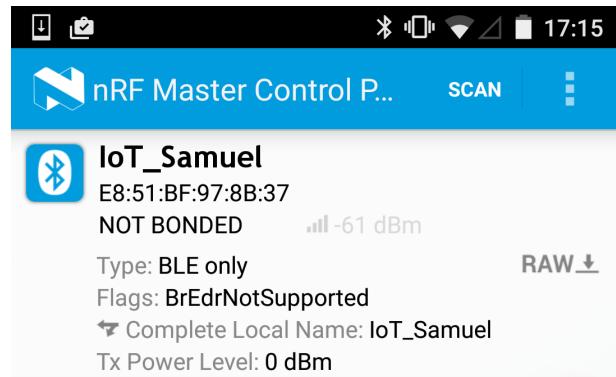
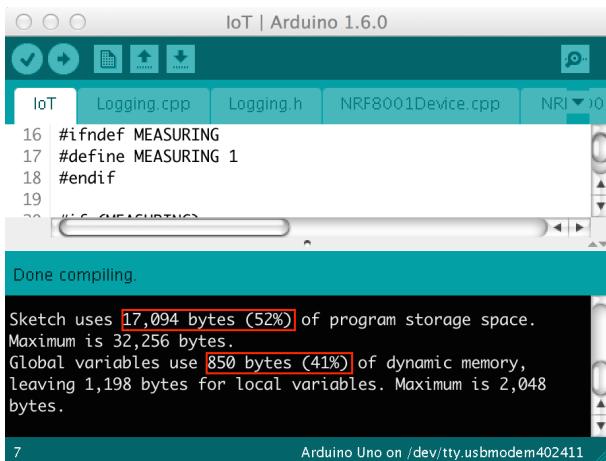


Figure 52: Liste des périphériques BLE en mode « broadcast »

Finalement, suite à la compilation du programme, nous pouvons attester du bon fonctionnement des macros, servant à « isoler » le code dédié aux mesures », observant la taille du sketch et la mémoire dédiée aux variables, en fonction de l'activation ou non des mesures (« MEASURING » à 1 ou à 0).

- « MEASURING » à 1 : taille du sketch : 17kb (52 %), taille des variables : 850 bytes (41%)
- « MEASURING » à 0 : taille du sketch : 14kb (43%), taille des variables : 810 bytes (39%)



The screenshot shows the Arduino IDE interface with the title bar "IoT | Arduino 1.6.0". The code editor contains the following code:

```

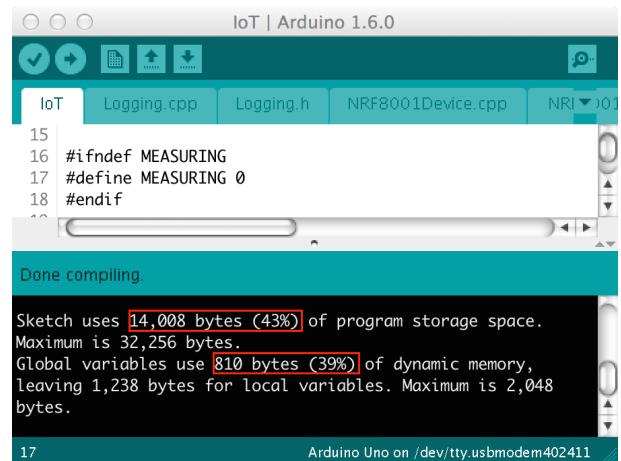
16 #ifndef MEASURING
17 #define MEASURING 1
18 #endif
19
20 //<-- MEASURING

```

The status bar at the bottom indicates "Done compiling." and "Arduino Uno on /dev/tty.usbmodem402411". Below the status bar, a message box displays memory usage:

Sketch uses 17,094 bytes (52%) of program storage space.
Maximum is 32,256 bytes.
Global variables use 850 bytes (41%) of dynamic memory,
leaving 1,198 bytes for local variables. Maximum is 2,048
bytes.

Figure 53: Taille du sketch et de la mémoire avec les mesures



The screenshot shows the Arduino IDE interface with the title bar "IoT | Arduino 1.6.0". The code editor contains the following code:

```

15
16 #ifndef MEASURING
17 #define MEASURING 0
18 #endif
19
20

```

The status bar at the bottom indicates "Done compiling." and "Arduino Uno on /dev/tty.usbmodem402411". Below the status bar, a message box displays memory usage:

Sketch uses 14,008 bytes (43%) of program storage space.
Maximum is 32,256 bytes.
Global variables use 810 bytes (39%) of dynamic memory,
leaving 1,238 bytes for local variables. Maximum is 2,048
bytes.

Figure 54: Taille du sketch et de la mémoire sans les mesures

Synthèse des tests réalisés :

- ✓ Importation des différents fichiers / librairie nécessaires au nRF8001
- ✓ Instanciation, initialisation et utilisation du nRF8001 en mode « Connect » et « Broadcast »
- ✓ Transformation du code précédent (prise de mesures) sous la forme de macros et lecture des capteurs lorsqu'un périphérique est connecté au nRF8001

7.4 TP04

Suite au démarrage de la station météo, nous pouvons nous connecter, dès à présent à celle-ci, à l'aide de l'application Android « nRF Master Control Panel ». Dès que nous sommes connectés, nous pouvons récupérer uniquement la température, l'humidité ou les deux, en ouvrant le(s) « PIPE » dédié(s) à cette / ces caractéristique(s).

La propriété « **INDICATE** » des caractéristiques indique que les opérations sont activées par le client, mais initialisées par le serveur, qui fournit ainsi un moyen permettant d'envoyer des données au client¹⁶.

Chaque caractéristique dispose d'un UUID propre. L'UUID « **0x2A1C** » caractérise une mesure de température (« Temperature Measurement »). Quant à l'UUID « **0x2A6F** », il devrait caractériser une mesure d'humidité, mais celui-ci n'étant pas reconnu par la norme, d'où la nomination « Unknown Characteristic ».

The screenshot shows the nRF Master Control Panel interface. At the top, it says "SAMeteo" and "Status: CONNECTED NOT BONDED". Below that, it lists two services: "Unknown Service" (UUID: 750e1809-52a1-1db2-1fc6-304e1c857657) and "Temperature Measurement" (UUID: 0x2A1C). The "Temperature Measurement" section is expanded, showing properties like "Properties: INDICATE" and "Value: 21,00°C". A red arrow points to this section with the label "Pipe ouvert". Below it, another section for "Unknown Characteristic" (UUID: 00002a6f-0000-1000-8000-000005f9b34) is shown, with properties like "Properties: INDICATE" and "Value: (0x) 00-2C". A red arrow points to this section with the label "Humidité".

Figure 55: Données issues de la station météo

```
/dev/cu.usbmodem402411
Send
INFO at time 0 h 3 m 10 s 476 ms: ACK for humidity measurement received
INFO at time 0 h 3 m 11 s 184 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x0000000f
DEBUG at time 0 h 3 m 11 s 407 ms: Evt Data credit: available 77
INFO at time 0 h 3 m 11 s 508 ms: ACK for temperature measurement received
DEBUG at time 0 h 3 m 12 s 291 ms: Evt Data credit: available 78
INFO at time 0 h 3 m 12 s 393 ms: ACK for temperature measurement received
DEBUG at time 0 h 3 m 14 s 477 ms: Evt Data credit: available 79
INFO at time 0 h 3 m 14 s 579 ms: ACK for temperature measurement received
DEBUG at time 0 h 3 m 14 s 690 ms: Evt Data credit: available 80
INFO at time 0 h 3 m 14 s 792 ms: ACK for humidity measurement received
DEBUG at time 0 h 3 m 16 s 573 ms: Evt Data credit: available 81
INFO at time 0 h 3 m 16 s 674 ms: ACK for humidity measurement received
DEBUG at time 0 h 3 m 18 s 656 ms: Evt Data credit: available 82
INFO at time 0 h 3 m 18 s 757 ms: ACK for temperature measurement received
DEBUG at time 0 h 3 m 18 s 868 ms: Evt Data credit: available 83
INFO at time 0 h 3 m 18 s 971 ms: ACK for humidity measurement received

Autoscroll No line ending 9600 baud
```

Figure 56: Aperçu sur la console des différents acquittements reçus suite aux envois de température et d'humidité

¹⁶ <http://mbientlab.com/blog/bluetooth-low-energy-introduction/>

En soufflant sur le capteur DHT11 ou en appuyant son doigt sur celui-ci, nous pouvons faire varier les valeurs lues par le capteur. Dès lors, de nouvelles données sont envoyées vers le Smartphone ; nous pouvons donc observer sur la capture ci-dessus, les acquittements reçus suite à ces envois.

Synthèse des tests réalisés :

- ✓ Connexion à la station météo
- ✓ Envoi de la température et / ou de l'humidité seulement si le PIPE est disponible et que la valeur à envoyer est différente de la valeur précédemment envoyée
- ✓ Récupération de la température et de l'humidité sur le Smartphone
- ✓ Acquittement des données envoyées

7.5 TP05

Dès que la station météo est alimentée, nous pouvons dès lors nous connecter. Nous pouvons récupérer uniquement la température, l'humidité ou la pression, en ouvrant le / les pipe(s) dédié(s) à cette / ces caractéristique(s).

Chaque caractéristique dispose d'un UUID propre. L'UUID « **0x2A6D** » caractérise une mesure de pression, mais celui-ci ne respectant pas la norme, il est donc dénommé « Unknown Characteristic ».

Données reçues :

- Température : 22 °C
- Humidité : 65 %
- Pression : 94393 Pa

Ensuite, afin de vérifier la bonne implémentation du thermostat, nous procédons à la modification de sa température minimale et maximale.

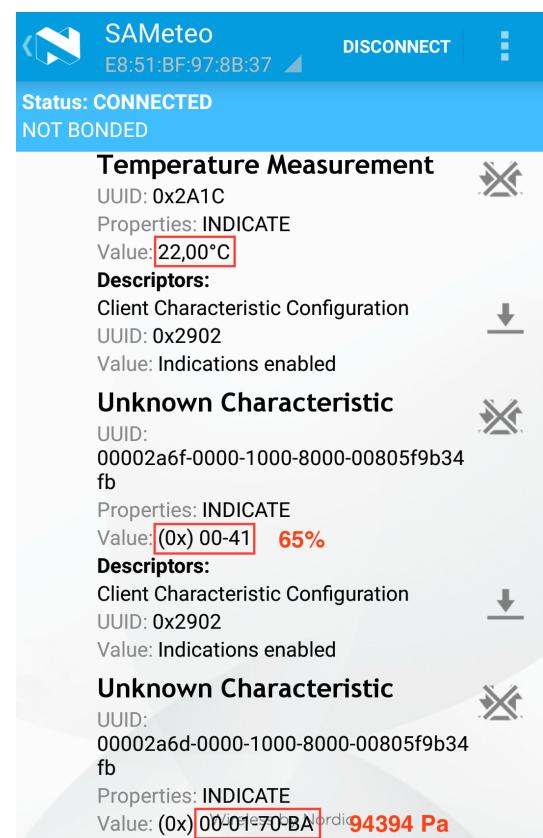


Figure 57: Données issues de la station météo

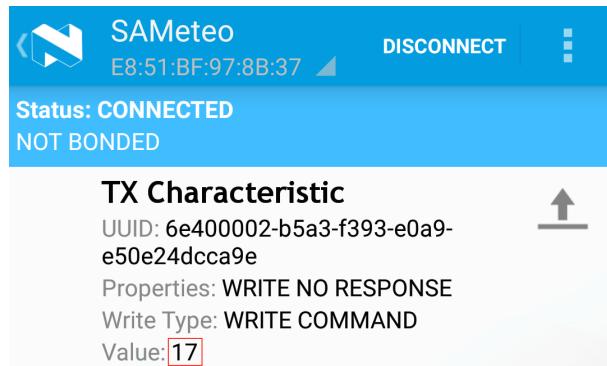


Figure 58: Modification de la température minimale

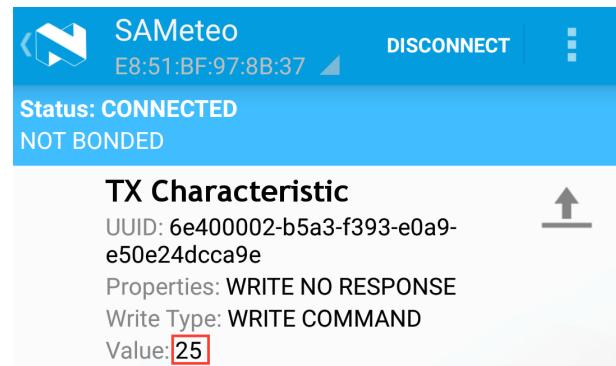


Figure 59: Modification de la température maximale

Etapes :

1. Ouverture du pipe dédié à la température
2. Réception de l'acquittement pour la température
3. Ouverture du pipe dédié à l'humidité
4. Réception de l'acquittement pour l'humidité
5. Ouverture du pipe dédié à la pression
6. Réception de l'acquittement pour la pression

```

INFO at time 0 h 1 m 58 s 111 ms: Broadcasting restored
INFO at time 0 h 1 m 58 s 347 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x00000007
INFO at time 0 h 2 m 1 s 936 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x0000000f 1
DEBUG at time 0 h 2 m 2 s 127 ms: Evt Data credit: available 3
INFO at time 0 h 2 m 2 s 163 ms: ACK for temperature measurement received 2
INFO at time 0 h 2 m 2 s 844 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x0000001f 3
DEBUG at time 0 h 2 m 3 s 69 ms: Evt Data credit: available 4
INFO at time 0 h 2 m 3 s 105 ms: ACK for humidity measurement received 4
INFO at time 0 h 2 m 3 s 748 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x0000003f 5
DEBUG at time 0 h 2 m 3 s 938 ms: Evt Data credit: available 5
INFO at time 0 h 2 m 3 s 974 ms: ACK for pressure measurement received 6

```

Autoscroll No line ending 9600 baud

Figure 60: Acquittement des données transmises

Etapes :

1. Simple pression sur le bouton-poussoir (→ activation du buzzer)
2. Double pression sur le bouton-poussoir (→ température max.)
3. Modification de la température maximale depuis le téléphone (25°C, cf. figure ci-dessus)
4. Double pression sur le bouton-poussoir (→ température min.)
5. Modification de la température minimale depuis le téléphone (17°C, cf. figure ci-dessus)
6. Simple pression sur le bouton-poussoir (→ désactivation du buzzer)

```

INFO at time 0 h 2 m 16 s 906 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x00000007
INFO at time 0 h 2 m 23 s 717 ms: Notification: buzzer enabled 1
INFO at time 0 h 2 m 23 s 718 ms: Button: One-click
INFO at time 0 h 2 m 33 s 541 ms: Button: Double-click 2
INFO at time 0 h 2 m 48 s 384 ms: Evt Data Received
INFO at time 0 h 2 m 48 s 385 ms: Thermostat max temperature set to : 25 3
INFO at time 0 h 2 m 53 s 564 ms: Button: Double-click 4
INFO at time 0 h 3 m 9 s 632 ms: Evt Data Received
INFO at time 0 h 3 m 9 s 633 ms: Thermostat min temperature set to : 17 5
INFO at time 0 h 3 m 24 s 526 ms: Button: One-click 6

```

Autoscroll No line ending 9600 baud

Figure 61: Modifications des paramètres liés au thermostat

Synthèse des tests réalisés :

- ✓ Ajout d'une caractéristique permettant l'envoi de la pression atmosphérique
- ✓ Gestion d'une pression simple ou double sur le bouton-poussoir
- ✓ Notifications réalisées à l'aide d'une LED RGB et d'un buzzer
- ✓ Activation / désactivation du buzzer (pression simple du bouton)
- ✓ Modification du thermostat (températures min. et max, pression double du bouton)

REMARQUE :

L'implémentation actuelle ne comporte aucun aspect lié à la sécurité. Des données sensibles (thermostat) étant transmises à la station, il serait plus que nécessaire de chiffrer la communication et de vérifier la cohérence des valeurs fournies.

7.6 TP06

Dans un premier temps, nous vérifions la bonne implémentation de la classe « Measurements » (partie I du TP), en envoyant, par intervalle de 5 secondes, une mesure à la classe à l'aide de la méthode « new_measurement(référence mesure, timestamp) ». Nous appelons ensuite les méthodes « get_nb_measurements() » et « get_last_measurement_index() » afin de nous retourner le nombre de mesures présentes dans le buffer, ainsi que l'index de la dernière mesure stockée.

Scénario :

- Une mesure de température / humidité est effectuée toutes les 5 secondes
- Le buffer peut accueillir 10 mesures
- Une mesure (au sens statistique : min / max / avg) dure 30 secondes

Nous pouvons constater sur la figure ci-jointe, le bon fonctionnement de la classe. En effet, lorsque le buffer est rempli, le nombre de mesures reste constamment à 10 (valeur de N), mais l'index continu de s'incrémenter par pas de 1, tout en restant dans l'intervalle du buffer [0-9].

Dans un second temps, nous vérifions la bonne implémentation de la partie II, dont une partie du code a été fourni par M. Ayer (cf. Moodle > nRFMeteoStation.cpp/.h). Après avoir installé l'application « nRF UART » sur le Nexus 5, nous procédons à des tests dans deux cas distincts :

- L'envoi de l'historique lorsque le buffer n'est pas plein (nombre de mesures inférieur à N)
- L'envoi de l'historique lorsque le buffer est plein (nombre de mesures égal à N)

Scénario :

- Une mesure de température / humidité est effectuée toutes les 5 secondes
- Le buffer peut accueillir 5 mesures
- Une mesure (au sens statistique : min / max / avg) dure 15 secondes

Buffer non plein :

Etapes :

1. Connexion à la station météo à l'aide de l'application « nRF UART »
2. Elaboration des statistiques pour la première période de mesures

```

INFO at time 0 h 0 m 20 s 814 ms: Evt Connected
INFO at time 0 h 0 m 21 s 16 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x00000007
INFO at time 0 h 0 m 28 s 913 ms: Thermostat max temperature set to : 29
INFO at time 0 h 0 m 30 s 501 ms: Temperature: min / max / avg : 25 / 25 / 25
INFO at time 0 h 0 m 30 s 594 ms: Number of measure: 1 Last measurement index: 0
INFO at time 0 h 1 m 5 s 501 ms: Temperature: min / max / avg : 25 / 25 / 25
INFO at time 0 h 1 m 5 s 514 ms: Humidity: min / max / avg : 41 / 43 / 41
INFO at time 0 h 1 m 5 s 591 ms: Number of measure: 2 Last measurement index: 1
INFO at time 0 h 1 m 40 s 501 ms: Temperature: min / max / avg : 25 / 25 / 25
INFO at time 0 h 1 m 40 s 515 ms: Humidity: min / max / avg : 41 / 46 / 43
INFO at time 0 h 1 m 40 s 593 ms: Number of measure: 3 Last measurement index: 2
INFO at time 0 h 2 m 15 s 501 ms: Temperature: min / max / avg : 25 / 26 / 25
INFO at time 0 h 2 m 15 s 516 ms: Humidity: min / max / avg : 49 / 62 / 56
INFO at time 0 h 2 m 15 s 593 ms: Number of measure: 4 Last measurement index: 3
INFO at time 0 h 2 m 50 s 501 ms: Temperature: min / max / avg : 27 / 31 / 29
INFO at time 0 h 2 m 50 s 515 ms: Humidity: min / max / avg : 70 / 81 / 77
INFO at time 0 h 2 m 50 s 593 ms: Number of measure: 5 Last measurement index: 4
INFO at time 0 h 3 m 25 s 501 ms: Temperature: min / max / avg : 29 / 31 / 30
INFO at time 0 h 3 m 25 s 515 ms: Humidity: min / max / avg : 81 / 86 / 83
INFO at time 0 h 3 m 25 s 593 ms: Number of measure: 6 Last measurement index: 5
INFO at time 0 h 4 m 0 s 501 ms: Temperature: min / max / avg : 28 / 29 / 28
INFO at time 0 h 4 m 0 s 515 ms: Humidity: min / max / avg : 75 / 79 / 77
INFO at time 0 h 4 m 0 s 591 ms: Number of measure: 7 Last measurement index: 6
INFO at time 0 h 4 m 35 s 501 ms: Temperature: min / max / avg : 26 / 28 / 27
INFO at time 0 h 4 m 35 s 515 ms: Humidity: min / max / avg : 69 / 73 / 70
INFO at time 0 h 4 m 35 s 593 ms: Number of measure: 8 Last measurement index: 7
INFO at time 0 h 5 m 10 s 501 ms: Temperature: min / max / avg : 27 / 27 / 27
INFO at time 0 h 5 m 10 s 516 ms: Humidity: min / max / avg : 63 / 67 / 65
INFO at time 0 h 5 m 10 s 593 ms: Number of measure: 9 Last measurement index: 8
INFO at time 0 h 5 m 45 s 501 ms: Temperature: min / max / avg : 26 / 30 / 27
INFO at time 0 h 5 m 45 s 516 ms: Humidity: min / max / avg : 58 / 61 / 60
INFO at time 0 h 5 m 45 s 593 ms: Number of measure: 10 Last measurement index: 9
INFO at time 0 h 6 m 20 s 502 ms: Temperature: min / max / avg : 26 / 26 / 26
INFO at time 0 h 6 m 20 s 516 ms: Humidity: min / max / avg : 56 / 58 / 57
INFO at time 0 h 6 m 20 s 594 ms: Number of measure: 10 Last measurement index: 0
INFO at time 0 h 6 m 55 s 501 ms: Temperature: min / max / avg : 25 / 26 / 25
INFO at time 0 h 6 m 55 s 515 ms: Humidity: min / max / avg : 53 / 56 / 54
INFO at time 0 h 6 m 55 s 593 ms: Number of measure: 10 Last measurement index: 1

```

Figure 62: Vérification du bon fonctionnement de la classe « Measurements »

Buffer plein :

Etapes :

1. Connexion à la station météo à l'aide de l'application « nRF UART »
2. Elaboration des statistiques pour la première période de mesures

3. Elaboration des statistiques pour la seconde période de mesures
4. Envoi du mot clé « send » depuis l'application mobile
5. La station indique qu'il y a deux mesures à transmettre (effectivement, deux mesures sont stockées dans le buffer ; cf. pt 3)
6. Envoi de la plus ancienne des mesures (index 0)

Remarque :

Nous pouvons remarquer que seulement la plus ancienne des mesures est transmise vers le téléphone mobile. La mesure stockée à l'index 1 n'a pas été transmise.

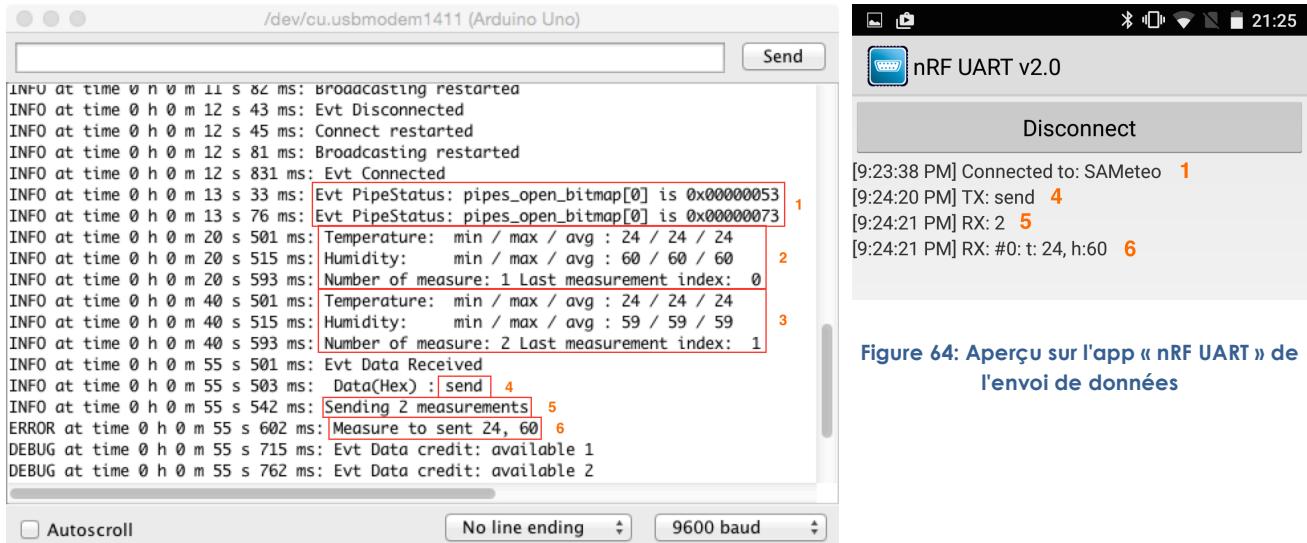


Figure 63: Aperçu sur la console lors de l'envoi de données

Figure 64: Aperçu sur l'app « nRF UART » de l'envoi de données

Buffer plein :

Etapes :

1. Elaboration des statistiques pour la période de mesures N + 1 (buffer circulaire ; écrasement des valeurs précédentes)
2. Envoi du mot clé « send » depuis l'application mobile
3. La station indique qu'il y a cinq (N) mesures à transmettre (effectivement, cinq mesures sont stockées dans le buffer ; cf. pt 1)
4. Envoi de la plus ancienne des mesures (index 0)
5. Réception de la mesure la plus ancienne dans l'application « nRF UART » (index 2 ; le « last measurement index » étant à 1)

Remarque :

Nous pouvons à nouveau remarquer un problème, seule la plus ancienne des mesures est transmise vers le téléphone mobile (index 2). Les mesures stockées aux cases 3 à N -1, ainsi que 0 à 1 n'ont pas été transmises.

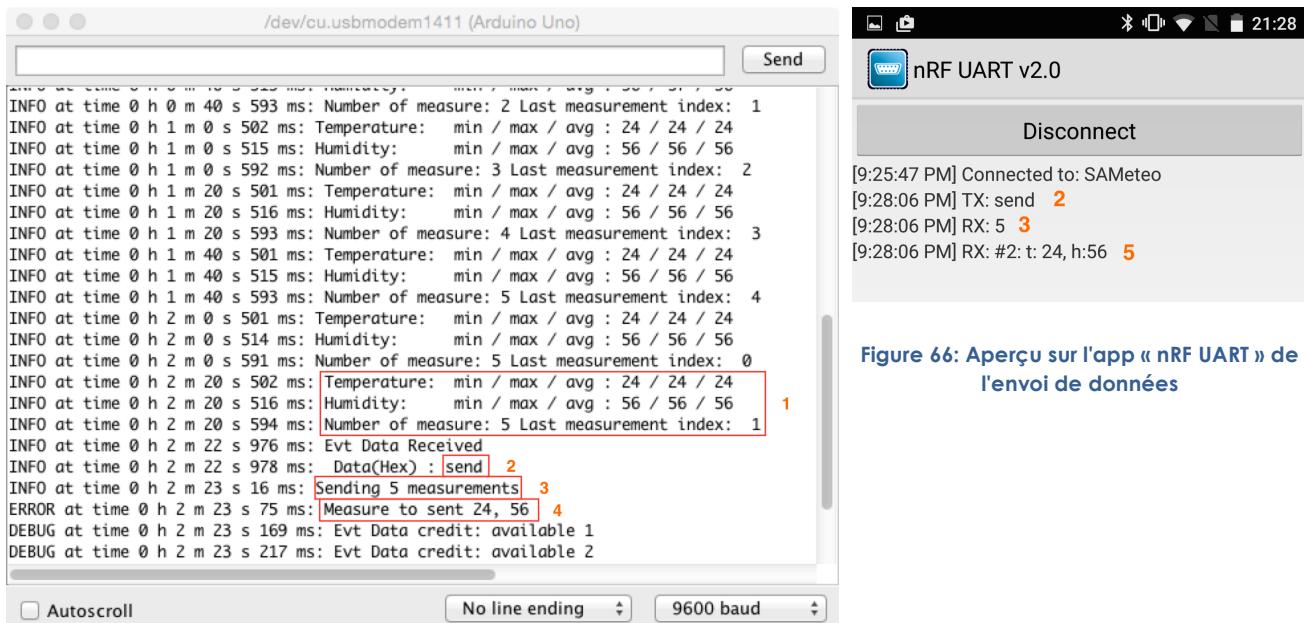


Figure 65: Aperçu sur la console lors de l'envoi de données

Figure 66: Aperçu sur l'app « nRF UART » de l'envoi de données

Pourtant, une modification a été apportée à l'instruction qui détermine l'index de la mesure à transmettre. En effet, le code est valide pour calculer la position de la plus ancienne des mesures lorsque le buffer n'est pas plein, mais lorsque celui-ci est complet, l'instruction renvoyait un index négatif.

Code initial :

```
measurementIndex = m_last_measurement_index - m_nb_measurements_to_send;
```

Correction apportée :

```
uint8_t measurementIndex = 0;
measure_value measure = {0};

if (m_nb_measurements_to_send < N) {
    while ((m_nb_measurements_to_send > 0) && (aci_state.data_credit_available >= 1)) {
        // send the measurement from the oldest to the newest
        measurementIndex = m_last_measurement_index - m_nb_measurements_to_send + 1;
        ...
    }
} else {
    measurementIndex = m_last_measurement_index;
    while ((m_nb_measurements_to_send > 0) && (aci_state.data_credit_available >= 1)) {
        // send the measurement from the oldest to the newest
        measurementIndex = (measurementIndex + 1) % N;
        ...
    }
}
```

Pour terminer, nous inspectons le fonctionnement global de la station, en utilisant l'UART pour définir une température minimale et maximale pour le thermostat, ainsi que pour récupérer à nouveau l'historique des mesures.

Scénario :

- Une mesure de température / humidité est effectuée toutes les 5 secondes
- Le buffer peut accueillir 5 mesures
- Une mesure (au sens statistique : min / max / avg) dure 15 secondes

- Définition de la température minimale à 17 degrés
- Définition de la température maximale à 28 degrés

Etapes :

1. Envoi de la valeur 17 depuis l'application mobile → définit la température minimale à 17 °C. Double-pression sur le bouton-poussoir.
2. Envoi de la valeur 28 depuis l'application mobile → définit la température maximale à 28 °C.
3. Elaboration des statistiques pour la troisième période de mesures
4. Envoi du mot clé « send » depuis l'application mobile
5. La station indique qu'il y a trois mesures à transmettre (effectivement, trois mesures sont stockées dans le buffer ; cf. pt 3)
6. Envoi de la plus ancienne des mesures (index 0)

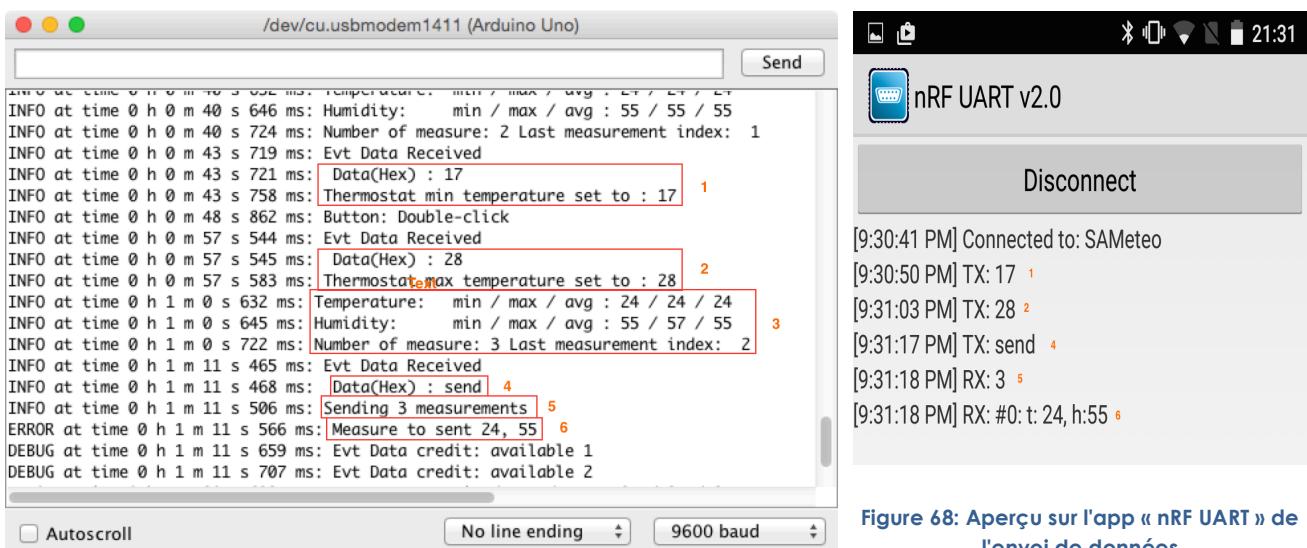


Figure 67: Aperçu sur la console lors de l'envoi de données

Figure 68: Aperçu sur l'app « nRF UART » de l'envoi de données

Après ces plus amples tests, nous pouvons affirmer que le fonctionnement du thermostat est toujours fonctionnel. Cependant, un problème persiste quant à l'envoi de l'entier de l'historique vers le Smartphone. D'autre part, si l'utilisateur envoie deux fois le mot clé « send », il ne se passera rien lors du second essai.

Synthèse des tests réalisés :

- ✓ Crédit de création de statistiques pour une période donnée (min / max / avg)
- ✓ Accesseurs pour accéder au nombre de mesures, à l'index du buffer ou pour récupérer une mesure spécifique
- ✓ Fonctionnement du thermostat via l'UART
- ✓ (partiellement) Récupération de l'historique des données stockée sur la station

8 PROBLEMES RENCONTRES & SOLUTIONS

13/03/2015 : Rendu du TP01 :

- J'ai rencontré quelques difficultés à comprendre le fonctionnement des macros de logging, mais après plusieurs explications, tout est rentré dans l'ordre.

25/03/2015 : Rendu du TP02 :

- Je n'ai pas rencontré de soucis à implémenter les tâches relatives au TP02.

17/04/2015 : Rendu du TP03 :

- Je n'ai pas rencontré de soucis à implémenter les tâches relatives au TP03.

12/05/2015 : Rendu du TP04 :

- J'ai rencontré quelques soucis à faire fonctionner l'envoi des données entre l'Arduino et le smartphone ; en feuilletant la présentation de Nordic (Moodle > Documentation and software > nRF8001), j'ai trouvé les solutions à mes problèmes. Quant à l'acquittement des envois, vos explications complémentaires, ainsi que les quelques exemples proposés par la librairie BLE, m'ont sorti de l'impasse.

28/05/2015 : Rendu du TP05 :

- Je n'ai pas rencontré de soucis à implémenter les tâches relatives au TP05.

15/06/2015 : Rendu du TP06 :

- Je n'ai pas rencontré de soucis quant à l'implémentation de la partie 1 du TP. Cependant, je rentre encore et toujours des difficultés à transmettre l'entier de l'historique des données ; seule la plus ancienne des mesures est transmises.

9 ACQUIS

TP01 :

- Ce premier travail pratique a été l'occasion d'apprivoiser la plateforme Arduino et de prendre en mains son IDE. D'autre part, il nous a permis de comprendre le fonctionnement de macros de logging et d'appréhender le fonctionnement de la mémoire sur l'Arduino.

TP02 :

- Ce second travail pratique a été l'occasion d'installer une librairie tierce et de l'appréhender, afin de pouvoir utiliser le capteur DHT11. D'autre part, ce travail m'a permis d'approfondir mes connaissances en électricité, en étudiant le fonctionnement des diviseurs de tension et des résistances « pullup ». Par curiosité, je me suis intéressé au Timer Watchdog, permettant de réduire la consommation de la carte Arduino.

TP03 :

- Ce troisième travail pratique nous a permis d'appréhender le fonctionnement de l'ACI et de ses différents évènements et commandes permettant la communication avec la puce nRF8001. D'autre part, nous avons été amenés à s'imprégnés d'une classe C++ tierce (« nRF8001Device »), d'en comprendre le fonctionnement et d'en utiliser les principales méthodes.

TP04 :

- Ce quatrième travail pratique nous a permis d'implémenter des concepts liés à l'ACI, en développant des fonctions permettant d'envoyer des données et de récupérer les acquittements de ces envois.

TP05 :

- Bien que ce cinquième TP repose sur des concepts appréhendés au TP précédents, ce fut l'occasion de mettre en place une caractéristique permettant de communiquer du Smartphone vers la station, afin de pouvoir définir la température minimale et maximale du thermostat.

TP06 :

- Ce dernier TP nous a permis de mettre en place un mécanisme permettant de stocker l'historique des mesures réalisées durant les N dernières heures, en termes de valeurs minimales, maximales ou moyennes.

10 PERSPECTIVES

Cette première approche avec l'Arduino laisse entrevoir de belles opportunités, en y branchant, par exemple, des capteurs afin de démarrer notre projet de station météo connectée.

Cette seconde approche, plus concrète, nous a permis de récolter des données réelles de température, d'humidité et de luminosité. Ces données pourront être traitées et consultées dans le futur, suite aux prochains travaux pratiques.

Cette troisième approche, très théorique et orientée principalement sur la communication entre l'ACI et la puce nRF8001, nous a permis d'appréhender la technologie Bluetooth. Tout nous laisse à penser, que dans des travaux futurs, nous serons amenés à réaliser les différentes fonctions nécessaires aux transferts de données entre la puce et les différents périphériques.

Cette quatrième approche, très pratique, nous a permis d'implémenter des fonctions liées à l'envoi des données. Nous pourrons donc, dans un futur proche, ajouter d'autres capteurs, sans difficultés, ni grands changements à l'intérieur du code.

Cette cinquième approche, très proche de ce qui avait été réalisé précédemment, laisse entrevoir la possibilité de stocker les données relatives aux capteurs sur une certaine durée afin de pouvoir réaliser des statistiques, que l'on pourra par la suite, transmettre plus loin.

11 CONCLUSION

Ce second travail fût intéressant, car il nous a permis de lier des concepts théoriques (diviseur de tension, résistance « pullup », etc) à des actes plus pratiques, comme la lecture des différents capteurs. Cela nous laisse donc de belles perspectives devant nous, à commencer par l'implémentation d'une communication sans-fil (Bluetooth) et à appréhender l'aspect de la consommation énergétique.

Ce quatrième travail, bien que séparé en deux parties, a été très conséquent. La première partie, reposant sur l'analyse des paquets échangés, fut bénéfique à la compréhension du fonctionnement des échanges, des connexions et des transmissions de données. La seconde partie, plus pratique, fut plus gratifiante car nous avons pu, pour la première fois, transmettre les données lues par les capteurs de la station météo vers le Smartphone.

Ce cinquième travail pratique fut l'occasion d'ajouter une nouvelle mesure, la pression atmosphérique. De plus, pour la partie libre de ce projet, nous avons pu implémenter toute la partie logique pour gérer une pression simple ou double sur un bouton, ainsi qu'ajouter une nouvelle caractéristique, permettant de définir depuis le téléphone la température minimale et maximale du thermostat.

Ce sixième TP met un point final à notre projet. Ce projet fut très intéressant car il nous a permis d'implémenter quelque chose de concret, une station météo connectée. D'autre part, ce fut l'occasion d'appréhender la plateforme Arduino, de se familiariser avec quelques notions d'électricité et la norme Bluetooth et d'employer un nouveau langage de programmation, C++.

12 ANNEXE

Le code source du projet se trouve sur Git, à l'adresse : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project>.

- TP01 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp01>
- TP02 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp02>
- TP03 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp03>
- TP04 : https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp04_I
- TP04 : https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp04_II
- TP05 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp05>
- TP06 : https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp06_II

13 REFERENCES

- <http://www.louisreynier.com/fichiers/KesacoArduino.pdf>
- <http://openclassrooms.com> (cours sur l'Arduino, plus disponible)
- <http://arduino.cc/en/Reference/Serial>
- <http://playground.arduino.cc/Code/AvailableMemory>
- <http://www.nongnu.org/avr-libc/user-manual/malloc.html>
- <http://arduino.cc/en/reference/map>
- Photorésistance : <http://www.instructables.com/id/Photocell-tutorial/?ALLSTEPS>
- Résistance « pullup » : <https://learn.sparkfun.com/tutorials/pull-up-resistors>
- Diviseur de tension : <https://learn.sparkfun.com/tutorials/voltage-dividers>
- DHT11 : <http://www.micropik.com/PDF/dht11.pdf>
- nRF8001 :
https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/24946618