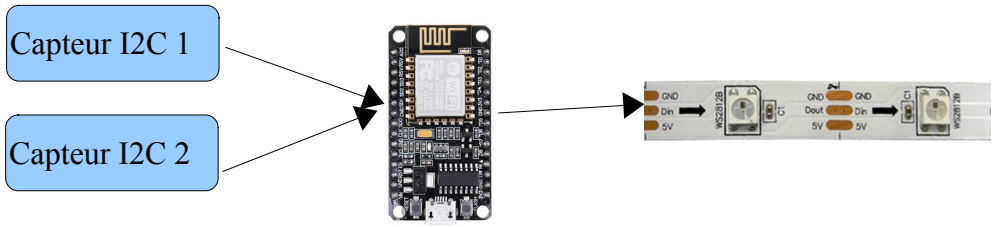
IOT S4 2021
Objet connecté

1. Présentation

Dans ce tp nous allons réaliser un objet connecté de démonstration basé sur un arduino NODEMCU ESP12E.
Cet objet comportera plusieurs capteurs (pas forcément identiques pour chaque binôme) reliés par bus I2C et une bande de led RGB "intelligentes" qui fera office d'"actionneur".
Nous pourrons utiliser un serveur MQTT pour connaître les valeurs mesurées par les capteurs et commander la bande de led. Un serveur HTTP pourra être programmé sur l'objet pour pouvoir y accéder directement.



Nous allons utiliser une plaque de câblage rapide.
Attention :

Ne pas modifier le câblage sous tension car il est très facile de se tromper de broche.
Toujours débrancher le connecteur USB avant de modifier le câblage.
Respectez les couleurs rouge pour le 3.3V et noir pour la masse

2. Essais de la bande de leds

Attention : un erreur de polarité détruit immédiatement toutes les leds de la bande.

Les leds RGB "chaînables" peuvent être associées en bande ou en panneaux et peuvent être pilotées par une seule broche. On peut donc commander un grand nombre de leds avec une seule sortie de l'arduino. On trouve facilement des bandes de plus de 100 leds.
La bande de led sera alimenté en 3.3V.
La broche **D8** de l'ESP est utilisée pour les données (**DATA_PIN**).

- Câblez la bande de led en faisant bien attention à la polarité.
- Installer la librairie FastLED : https://github.com/FastLED/FastLED
La documentation de cette libraire est ici : http://fastled.io/

- Essayez d'abord avec l'exemple *FastLed → Blink*
Pour l'ESP8266 plusieurs numérotations des broches sont possibles (ex D8=GPIO15).
Il faut définir la numérotation utilisée AVANT l'inclusion de l'entête.
Pour utiliser les numéros de GPIO (ex 15) ou les définitions équivalentes (ex D8) :

```
#define FASTLED_ESP8266_RAW_PIN_ORDER
#include <FastLED.h>
```

Il faut modifier les constantes :

NUM_LEDS pour mettre le nombre de led de votre bande,
DATA_PIN pour mettre le numéro de la broche utilisée (D8 ou 15 sur notre carte).
(CLOCK_PIN n'est pas utilisée par nos leds).

Dans le `setup` ne garder que la ligne qui correspond à votre type de leds et commenter toutes les autres. Pour le type WS2812B il faut donc laisser :

```
FastLED.addLeds<WS2812B, DATA_PIN, RGB>(leds, NUM_LEDS);
```

Ordre des couleurs

Si tout est correct, la première led de la bande doit clignoter.
Il faut maintenant vérifier l'ordre des couleurs. On peut le modifier dans la ligne précédente `RGB,GRB,BGR,…` Avec l'exemple la première led doit s'allumer en rouge. Si ce n'est pas le cas essayez d'autres places pour `R` jusqu'à obtenir du rouge.
Ensuite il faut trouver la place des autres couleurs.
Pour cela modifiez la ligne :

```
leds[0] = CRGB::Red;
```

pour allumer en bleu :

```
leds[0] = CRGB::Blue;
```

Si on obtient pas du bleu, changez la place du `B` pour l'obtenir (sans changer la place du `R`!).
La configuration est maintenant correcte.
Vous pouvez vérifier que le vert est obtenu par `CRGB::Green`.

Pour modifier la luminosité de toutes les leds de la bande vous pouvez utiliser la commande suivante :

```
FastLED.setBrightness(64); // paramètre possible de 0 à 255
```

- Sauvegardez cet exemple modifié pour pouvoir y revenir en cas de doute.

Une fois le code correctement configuré, l'utilisation basique de la librairie est très simple :
On a un tableau de "couleurs" de la taille de la bande de led :

```
CRGB leds[NUM_LEDS];
```

La couleur de la première led est dans la case 0 , la deuxième dans la case 1, ….
Le tableau est appliqué sur la bande par l'instruction : `FastLED.show();`
La couleur est une variable de type `CRGB` (ou **CHSV** voir plus loin).
Par exemple pour allumer toute la bande on peut faire :

```
int i ;
CRGB rgb;
rgb.red=230;
rgb.green= 70;
rgb.blue= 24;
for(i=0; i<NUM_LEDS ; i++) {
      leds[i] = rgb ;
}
FastLED.show();
```

Ici `FastLED.show()` est placé *après* la boucle donc toutes les leds s'allument en même temps : le tableau est d'abord rempli puis appliqué sur la bande de leds.

Si on place `FastLED.show()` dans la boucle, les leds s'allument au rythme de la boucle for :

```
for(i=0; i<NUM_LEDS ; i++) {
      leds[i] = rgb ;
      FastLED.show();
      delay(200) ;
}
```

Si on veut faire varier la couleur de chaque led, on peut modifier la variable rgb dans la boucle. Par exemple ci-dessous le niveau de bleu augmente en s'éloignant de la première led :

```
for(i=0; i<NUM_LEDS ; i++) {
      rgb.blue = 10 * i ;
      leds[i] = rgb ;
      FastLED.show();
      delay(200) ;
}
```

Pour modifier la luminosité de toutes les leds de la bande vous pouvez utiliser la commande suivante :

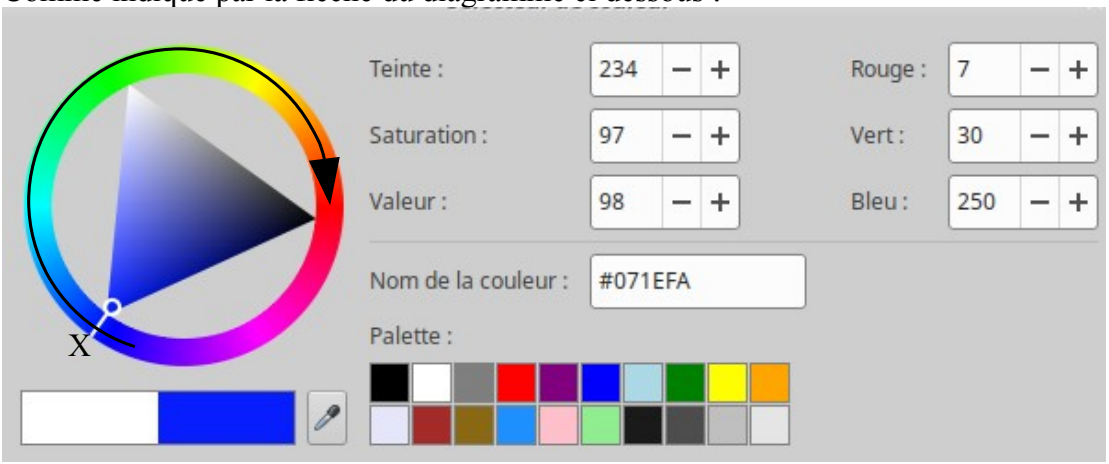
```
FastLED.setBrightness(64); // paramètre possible de 0 à 255
```

Voir les nombreuses autres possibilités dans les exemples et sur le site : http://fastled.io/

Ici nous avons utilisé l'attente bloquante avec la fonction `delay()` mais il ne faut pas l'utiliser par la suite lorsqu'on va intégrer des animations dans des programmes utilisant mqtt et/ou un serveur web car il ne faut pas diminuer le temps de réponse de loop().

3. Gestion de la couleur

Il existe différentes manières de décrire une couleur. On parle d'*espaces de couleurs*.
Selon l'application certains espaces sont plus adaptées que d'autre.
L'espace courant lorsqu'on pilote des leds est l'espace RGB. Il apparaît assez naturel mais n'est pas bien adapté lorsqu'on veut faire varier une couleur en fonction d'un paramètre scalaire (température, pression…) parce que la couleur est exprimée par les valeurs relatives de trois valeurs R,G et B. De plus l'intensité, qui dépend des valeurs absolues de R,G et B, est couplée à la couleur.
Il est donc judicieux d'utiliser une autre représentation de la couleur : l'espace HSV.
Par exemple on veut passer de bleu au rouge pour une température variant entre 0°C et 30°C.
Comme indiqué par la flèche du diagramme ci dessous :



Il existe de nombreux sites qui permettent de déterminer les valeurs RGB et HSV d'une couleur. Par exemple : https://colorpicker.me

Par exemple pour le bleu indiqué par le X, il faut 7/255 de rouge, 30/255 de vert et 250/255 de bleu. Pour faire varier la couleur en restant dans la représentation RGB il faudrait faire varier simultanément les trois valeurs (r,g,b).
Il est plus commode d'utiliser la représentation HSV.
Dans la représentation HSV, la couleur est portée par une seule composante **H** (hue=teinte).
S est la saturation : pureté de la couleur
V est le niveau de luminosité.
Ces deux derniers paramètres pourront rester constants dans notre application.
Attention selon les auteurs et les sites les intervalles peuvent être différents. On trouve :
pour H : [0,1] ,[0,100%], [0,360°] ou [0,255]
pour S : [0,1] , [0,100%] ou [0,255]
pour V : [0,1] , [0,100%] ou [0,255]

Par exemple la couleur bleu (X) précédente est représentée en HSV par H=234, S=97,V=98.
Ici Hmax=360, Smax=100, Vmax=100 donc H=234/360, S=97/100 et V=98/100.

Il est donc commode de travailler avec HSV puis si nécessaire avec une librairie comme **FastLED** nous allons disposer d'une fonction pour convertir HSV vers RGB[1].

On dispose de deux nouveaux "types" **CHSV** et **CRGB** pour manipuler la couleur et d'une fonction, `hsv2rgb_rainbow` , pour passer de HSV vers RGB.

1 La libraire **FastLED** permet de piloter directement les leds WS2812 avec le type HSV. La conversion est par contre nécessaire pour des leds RGB classiques qui se commandent séparément en pwm.

Le programme ci-dessous montre l'utilisation en fixant la couleur de départ bleu en H,S,V et en la faisant varier jusqu'au rouge comme indiqué par la flèche sur l'image précédente. Ici les valeurs H,S,V sont toutes des entiers dans l'intervalle [0,255].

```
#include <FastLED.h>
#define FASTLED_ESP8266_RAW_PIN_ORDER
#define DATA_PIN 15
CRGB leds[NUM_LEDS];

CHSV hsv; // "variable" contenant la couleur(hue,sat,value)

void setup() {
 FastLED.addLeds<WS2812B, DATA_PIN, GRB>(leds, NUM_LEDS);
 FastLED.setBrightness(16);
 hsv.value = 255;
 hsv.sat = 255 ;
 hsv.hue = 255 * (234 / 360.0) ; // bleu h=234
 for (int i = 0; i < NUM_LEDS; i++) {
      leds[i] = hsv;
 }
 FastLED.show();
}

void loop() {
 static unsigned long previous = 0 ;
 unsigned long now = millis();
 if ( now - previous > 20) {
 previous = now ;
 if ( hsv.hue > 0 ) {
      hsv.hue -= 1 ;  // la teinte diminue vers le rouge h=0
 }
 else {
      hsv.hue = 255 * (234 / 360.0) ; // puis repart du bleu h=234
 }
 for (int i = 0; i < NUM_LEDS; i++) {
      leds[i] = hsv;
 }
 FastLED.show();
 }
}
```

Maintenant vous devriez être capable de faire varier la couleur de la led en fonction d'une quantité scalaire en appliquant simplement une relation linéaire.

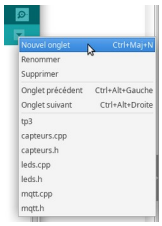
4. Un peu d'organisation…

On veut créer un programme assez élaboré avec différentes fonctionnalités : leds, capteurs, mqtt, serveur web,….
Il faut donc un minimum d'organisation pour s'y retrouver.
De manière à bien séparer les différentes fonctionnalités, on les codera dans des fichiers différents. Un fichier principal qui est le seul à porter l'extension **ino** appellera les initialisations (*setup_xxx()*) et les fonctions *loop_xxx()* de tous les autres :
Par exemple les fonctionnalités propres à la bande de leds sont dans led.cpp. Un fichier led.h contient les prototypes des fonctions à exporter.

Le programme aura donc la structure suivante :

tp3
- capteurs.cpp # contient les fonctions setup_capteurs(), loop_capteurs() et d'autres fonctions nécessaires par lire les capteurs.
- capteurs.h # contient les prototypes des fonctions de capteurs.cpp et des définitions de constantes.
- leds.cpp
- leds.h
- mqtt.cpp
- mqtt.h
- **tp3.ino** #programme "principal" contenant les fonctions setup et loop et portant le même nom que le répertoire.

Dans l'ide arduino vous pouvez utiliser la flèche à droite pour créer un nouveau fichier par la fonction "Nouvel onglet" :



Le fichier principal tp3.ino appelle les différentes fonctionnalités :

```
#include "leds.h"
#include "capteurs.h"
#include "mqtt.h"

void setup() {
 Serial.begin(115200);
 setup_leds();
 setup_capteurs();
 setup_mqtt();
}

void loop() {
 loop_leds();
 loop_capteurs();
 loop_mqtt();
}
```

Les fichiers .h ont la structure suivante :

```
#ifndef LEDS_H
#define LEDS_H

void setup_leds();
void loop_leds();

#endif
```

5. Leds et MQTT

- Créer votre programme principal, et créez les fichiers leds.h et led.cpp.
- Initialisez les leds dans `setup_leds()`.
- Créer une fonction `test_leds()` qui exécute une animation de la bande de led.
  Attention l'animation doit être non bloquante (utilisez `millis` et des états pour les temporisations).
- ajouter les fichiers mqtt.h et mqtt.cpp.
- effectuer la connexion au point d'accès wifi et au serveur mqtt de la salle.
- afficher l'adresse MAC de votre esp.
- piloter la couleur de la bande de led par des topics mqtt :

topics[2] : `espgeiiXXYY/leds/n/hsv` message : `h,s,v`

où :

XXYY sont les deux derniers octets de l'adresse MAC de votre esp.
n est le numéro de la led (0-7).
h,s,v valeurs h, s, et v dans l'intervalle 0-255.

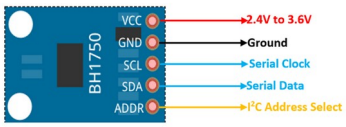
On veut pouvoir piloter la couleur de la première led par l'application Android MQTT Dash.

- Analysez le message envoyé par le widget "color" de l'application et créer un topic pour commander la première led par ce widget.

6. Création d'une librairie pour un capteur I2C BH1750

Le BH1750 est un capteur de luminosité avec une interface I2C. Bien qu'on trouve des libraires Arduino pour ce capteur, le but est d'apprendre à créer une libraire soit même à partir de l'étude de la documentation constructeur.

- Câbler le module capteur :



SCL : D1 GPIO5
SDA : D2 GPIO4
ADDR : GND ou +3.3V

D1 et D2 sont les broches utilisées par défaut pour l'I2C sur ESP8266. Il est possible de le changer lors de l'initialisation de la librairie Wire.

- Vérifier le câblage avec le programme I2Cscanner:
https://gist.github.com/tfeldmann/5411375
Au moins un circuit doit être trouvé.

6.1 Étude de la documentation constructeur

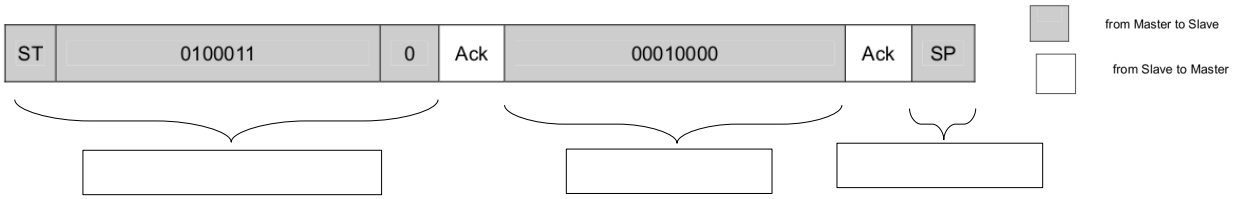
- Vérifier que l'adresse trouvée correspond bien à l'adresse déduite de la documentation.
- Décrire *très précisément* la suite de transactions I2C nécessaires pour :
  - Démarrer le circuit (PowerOn)
  - Faire un reset du circuit
  - Placer le circuit en mode *continu haute résolution*
  - Lire une mesure

2 Il faut donc s'abonner aux 8 topics : `espgeiiXXYY/leds/0/hsv, espgeiiXXYY/leds/1/hsv, espgeiiXXYY/leds/2/hsv,...`

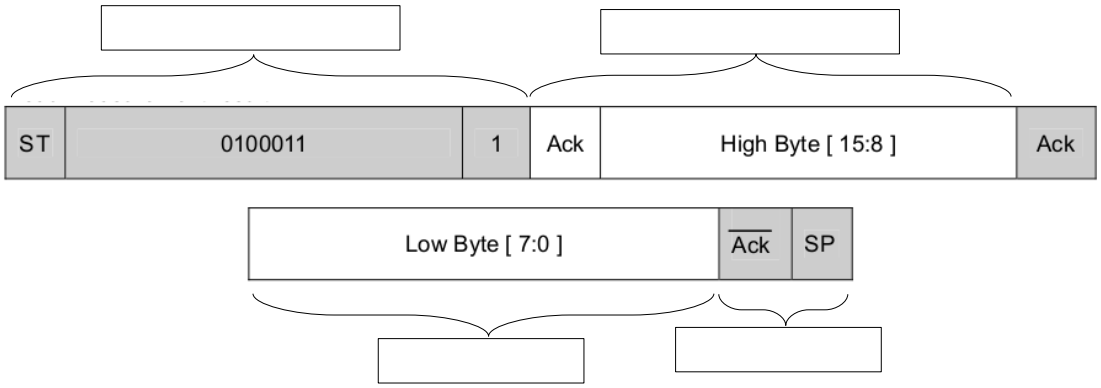
6.2 Étude de la librairie Wire

- En consultant la librairie arduino Wire[3]: https://www.arduino.cc/en/reference/wire , indiquer les appels de fonctions à utiliser pour effectuer les transactions ci dessous :

Écriture d'un octet :



Lecture de deux octets :



A quoi sert la méthode **`available()`** ? Quand doit on l'utiliser ?

6.3 Une première lecture de la luminosité

Travailler dans un programme différent du programme précédent. Une fois la librairie créée, il sera facile de l'intégrer au programme complet.

- En utilisant la librairie Wire, écrire un programme qui exécute les actions décrites en 6.1 pour lire et afficher la luminosité.

Comme il y a beaucoup d'actions identiques, il est judicieux d'écrire une fonction pour écrire un octet et un fonction pour lire les deux octets de la réponse.
Les prototypes seront :

```
void i2cWrite8( uint8_t address , uint8_t data ) ;

uint16_t i2cRead16( uint8_t address ) ;
```

- Coder ces fonctions et modifier votre programme pour les utiliser.

3 Pour l'ESP8266 la seule différence avec cette documentation est qu'on peut changer les broches utilisée par l'I2C voir : https://arduino-esp8266.readthedocs.io/en/latest/libraries.html#i2c-wire-library

6.4 Conception d'une classe C++ pour accéder au circuit BH1750

Le but est de disposer d'une classe C++ qui permet d'accéder facilement aux fonctionnalités du circuit sans avoir à connaître les détails de son fonctionnement.

La classe devra au moins comporter les membres suivant :

```
#ifndef BH1750_h
#define BH1750_h

#include "Arduino.h"

#include "Wire.h"

class BH1750 {
  public:
       BH1750(); // constructeur initialise l'adresse le mode par défaut
       void begin(void); // initialisation du circuit[4]
       void setAddress( uint8_t address ) ; // change l'adresse du circuit
       void setMode(uint8_t mode ); // change le mode de fonctionnement
       void reset(void); // reset du circuit
       uint16_t getRawLight(void); // lit le résultat brut d'une mesure
  private:
       void i2cWrite8( uint8_t data ) ; // écrit un octet sur le circuit
       uint16_t i2cRead16( ) ; // lit 16 bits sur le circuit
       uint8_t address ;
       uint8_t mode ;
 };
 #endif
```

Ensuite vous pouvez rajouter des méthodes pour utiliser les autres fonctionnalités du circuit et une méthode pour retourner la valeur de la luminosité en lux (la conversion est fonction du mode).

Vous pouvez partir du programme **BH1750_lib_ETD** fournit.

6.5 Intégration dans le système de libraires Arduino

- Dans le répertoire des librairies de l'IDE, créer un répertoire **LibBH1750**, et y placer les fichiers .cpp et .h.

- Créer un sous répertoire **examples** et y placer votre programme de test de la librairie.

Après avoir redémarré l'IDE, votre librairie est utilisable dans n'importe
que programme et votre exemple est directement accessible.

Vous pouvez rajouter un fichier **keywords.txt** pour avoir la coloration syntaxique (voir la syntaxe exacte ici : https://arduino.github.io/arduino-cli/latest/library-specification/).

4 Il ne faut pas initialiser la libraire wire ici car on peut avoir plusieurs capteurs i2c. Il faut laisser l'initialisation dans le setup principal.

7. Lumière et MQTT

- Reprenez votre programme complet, et créez les fichiers capteurs.h et capteurs.cpp.

- Initialisez le BH1750 dans `setup_capteurs()`

- Mesurez la luminosité tous les 2 secondes et publier la valeur sur le topic :
`espgeiiXXYY/light`

Comme la gestion du capteur et la partie mqtt sont dans des fichiers différents il faut rendre une variable globale déclarée dans un fichier accessible dans un autre :
- soit la luminosité[5] doit être visible dans le fichier mqtt.cpp
- soit le client mqtt doit être visible dans le fichier capteurs.cpp[6].

Une variable globale est exportée en répétant la déclaration (pas l'initialisation) dans le fichier dans lequel on veut y accéder.
Exemple :
La variable x globale dans f1.cpp :

```
int x=12 ;// déclaration et initialisation.
```

est rendue visible dans f2.cpp par :

```
extern int x ;  // on répète la déclaration seulement précédée de extern
```

Pour rendre visible le client mqtt dans capteurs.h :

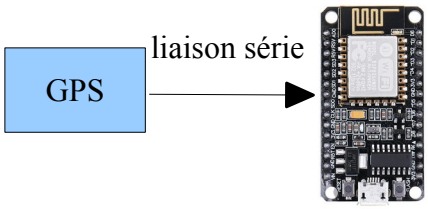
```
extern PubsubClient client ;
```

Les paramètres d'appels aux constructeurs ne sont pas répétés : il suffit de se rappeler que le compilateur a seulement besoin de connaître le type de la variable.

5 Ou l'objet représentant le capteur.
6 C'est plus cohérent ici : la publication mqtt pourra être facilement synchronisée avec la mesure.

8. Application : Alerte Antivol

En connectant un GPS à notre objet on peut lui faire émettre une alarme par un message MQTT ou localement (visuelle ou sonore) lorsqu'il sort d'une zone autorisée[7].



Comme nous n'avons qu'un seul port série accessible sur l'ESP12E, nous allons utiliser un librairie qui émule un port série : **`SoftwareSerial`**
Le principe est le suivant :
Au départ l'objet est configuré par un message mqtt pour lui donner sa position de référence et la diamètre de la zone autorisée.
Par exemple `{"position":[43.0275,06.4536],"dmax":500}`
Les trames GPS sont reçues sur le port série soft.
Le programme sélectionne les trames GGA uniquement.
Il calcule la distance au point de référence.
Si cette distance est dépassée il émet un message d'alarme et il émet périodiquement sa position.

8.1 Étude de l'exemple `Communication→SerialEvent`

Cet exemple montre comment former une chaîne String à partir des caractères reçus et jusqu'à rencontrer un caractère de fin ('\n' dans l'exemple).

Attention : l'exemple est prévu pour un arduino ATMEGA dans lequel la fonction serialEvent est appelée systématiquement après chaque appel à loop(). Pour le faire fonctionner avec l'ESP il faut appeler la fonction serialEvent dans la fonction loop().

- Soyez sûr de bien comprendre cet exemple.
- Modifier le pour ne pas inclure le caractère de fin (\n) dans la chaîne.

8.2 Adaptation à `SoftwareSerial`

- Câblez un adaptateur USB → TTL **3.3V :**
Seul les fils de masse et Tx sont nécessaires. Utiliser l'entrée **D3** de l'ESP.

Un nouveau port série apparaît. Pour émettre des caractères sur ce port il faut utiliser un émulateur de terminal (GTKterm, serialStudio, minicom,…) ou le terminal d'une autre instance de l'IDE arduino.

- En utilisant l'exemple EspSoftwareSerial→ swserTest, adaptez le programme précédent pour recevoir des caractères sur le port série USB→ TLL et afficher la trame sur le port série de l'ESP.

7 Ici la connexion au server MQTT est faite en Wifi. Ce n'est pas très réaliste, pour une application en extérieur on utilisera plutôt une connexion par GSM ou par un réseau LORA et une passerelle vers le serveur MQTT.

8.3 Utilisation d'un simulateur de GPS

Dans l'atelier le signal GPS n'est pas reçu et de toute façon il n'est pas envisageable d'avoir à déplacer l'objet pour faire les premiers essais.
Nous allons donc utiliser le simulateur de trame NMEA :

https://github.com/panaaj/nmeasimulator

- Installez le logiciel et configurez le pour émettre des trames sur le port série USB.

- Vérifiez que vous recevez correctement les trames émises par le logiciel.

- Utilisez les fonctions du tp précédent pour extraire les données de la trame GGA.

8.4 Intégration dans le programme général

- Reprenez votre programme général, et créez les fichiers gps.h et gps.cpp.

- Placez y le code pour recevoir et décoder les trames GPS.

- Faire l'initialisation de la position et de la distance par un message mqtt :

topic : `espgeiiXXYY/configGPS`
message : `{"position":[43.0275,06.4536],"dmax":500}`

(utiliser le format le plus pratique pour les coordonnées, la distance est en mètres).

Le format du message est appelé format **JSON**[8]. Vous pouvez le décoder "à la main" mais vous avez intérêt à utiliser la libraire ArduinoJSON : https://arduinojson.org/
Inspirez vous de l'exemple **JsonParserExemple**.

- Calculer la distance entre la position courante et la position de référence.

- Faire clignoter la led ANT (D4) si on sort de la zone autorisée.

- Tant qu'on est hors de la zone autorisée émettre un message avec la position et l'écart en m :

topic : `espgeiiXXYY/alerteGPS`
message : `{"position":[43.0385,06.4812],"ecart":1200}`

8 Voir par exemple : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation