Tutoriel MQTT Homeassistant & Ardiuno pour les nuls débutants (ou pour les nuls, c'est vous qui voyez)

Bonjour à toutes et à tous,

Je viens de passer quelques jours à m'amuser à trouver du code un peu partout sur internet pour pouvoir créer mes scripts et faire fonctionner mon Arduino et le coupler à mon HomeAssistant.

Il y a plusieurs manières d'écrire un code pour réaliser le même travail, du coup, pour ceux qui ne sont pas experts en C++, ça peut être compliqué de comprendre.

Avant de commencer, je voudrais préciser que je ne connais pas du tout C++, mais j'ai programmé en Basic et en Pascal il y a une vingtaine d'années. Autant dire que le tout est un peu rouillé !
De plus la barrière de la langue pour beaucoup de francophones peut être un souci.

So let's go 😉!

Matériel utilisé

- Un Raspberry 3B+ sur lequel est installé Homeassistant (en mode Home Assistant Operating System)
- Un Arduino Uno (Rev3)
- Un shield ethernet W5100 (Keyestudio ks0156)
- Une carte Elegoo "8 Channel Relay module"
- Une breadboard
- Une sonde Elegoo photo resistor (Photorésistance)
- Une (ou plusieurs) sonde(s) de température 1Wire (DS18B20)
- Une résistance de 4.7KOhm
- Cable Ethernet RJ45 (connexion au réseau)
- Cable Usb (alimentation)

Quelques photos du matériel

Figure 1 : Carte Arduino UNO

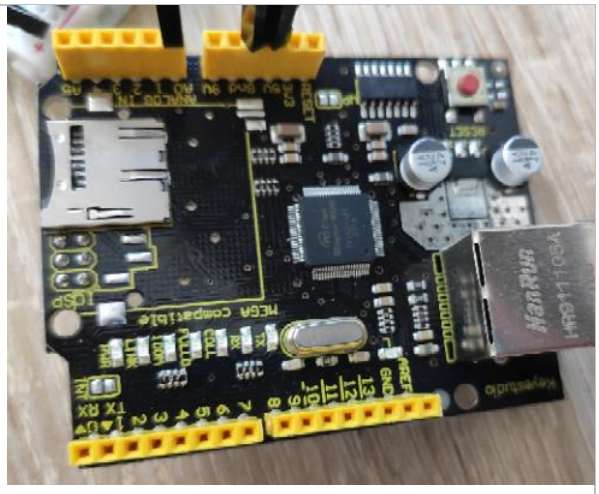


Figure 2 : Shield Ethernet W5100

Figure 3 : Carte Elegoo 8 relais

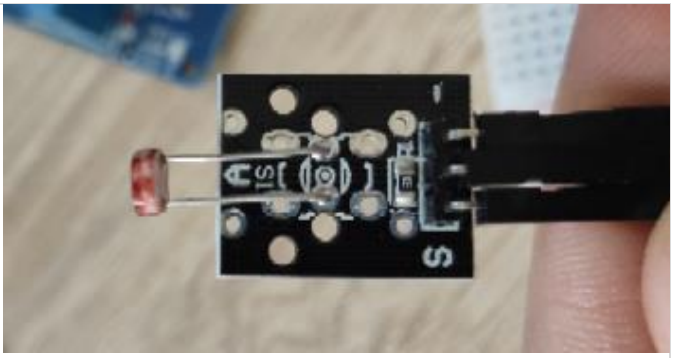


Figure 4 : La sonde photorésistance

Figure 5 : Une sonde DS18B20 étanche

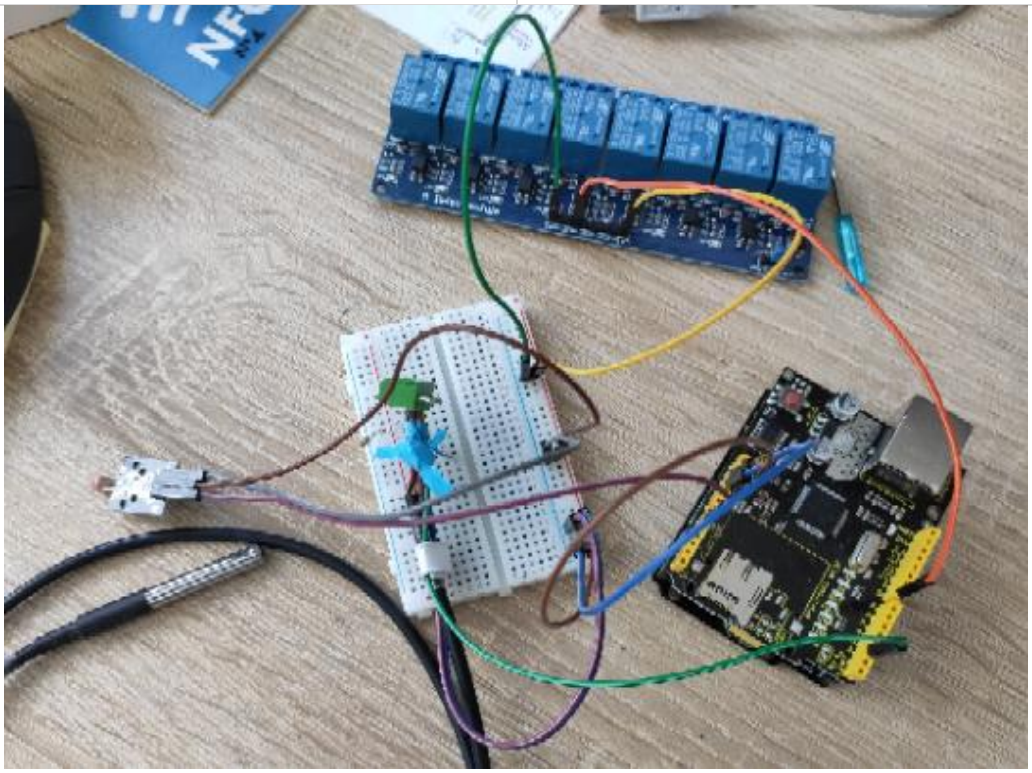


Figure 6 : Le montage complet

Le schéma de câblage

Désolé, j'ai pas trouvé les composants exacts sur Fritzing, de toute façon, vous lirez la notice de vos propres composants pour les branchements.

Pour rappel si besoin :
VCC = alimentation électrique (5 ou 3.3V)
Signal = Le retour du signal vers un port de l'arduino
Ground = Le Ground (le neutre)

Attention, vous aurez probablement remarqué que j'ai positionné 3 sondes de températures 1Wire DS18B20 sur le schéma et sur les photos il n'y en a qu'une seule.

Dans le code fourni, vous trouverez par contre, tout ce qu'il faut pour configurer plusieurs sondes.

L'avantage des sondes 1Wire, c'est qu'elles fonctionnent toutes sur le même bus. Ce qui veut dire que vous pouvez avoir plusieurs sondes branchées sur un seul port de l'arduino (le câble jaune qui rentre dans l'entrée Digital 2 de l'arduino sur le schéma ci-dessous). On peut aussi les brancher/débrancher à chaud. De plus elles ne sont pas chères (~5€ la sonde) et très précises (pas de 0.06°C).

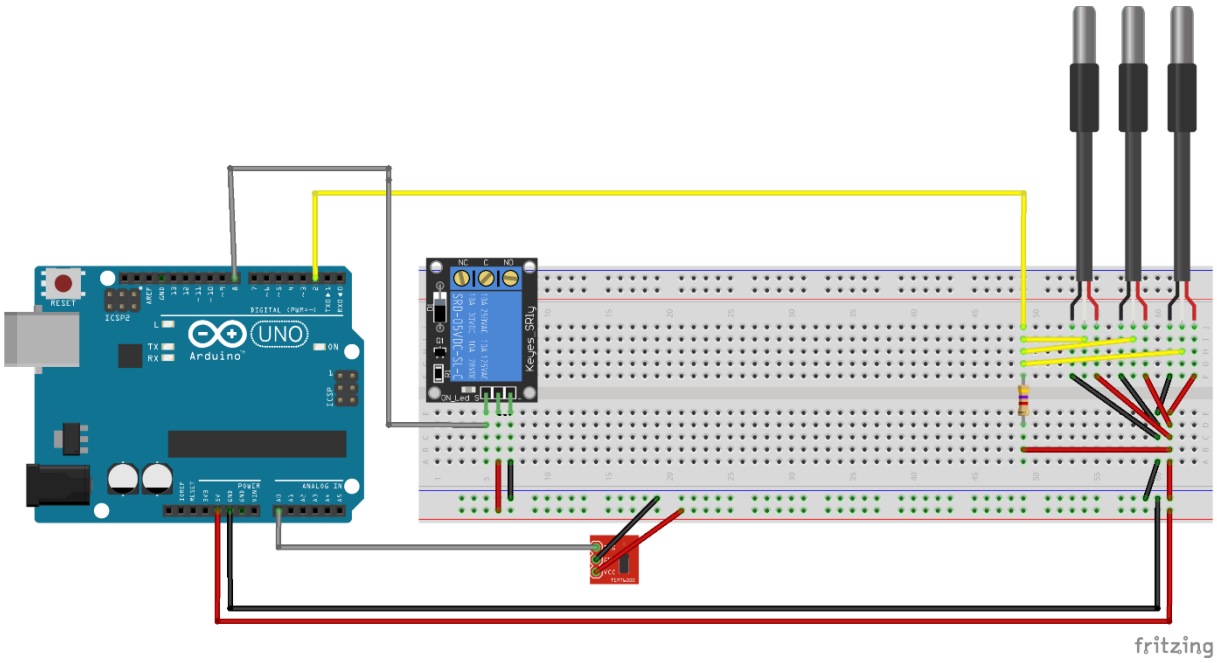


Figure 7 : Les branchements sur la platine d'essai

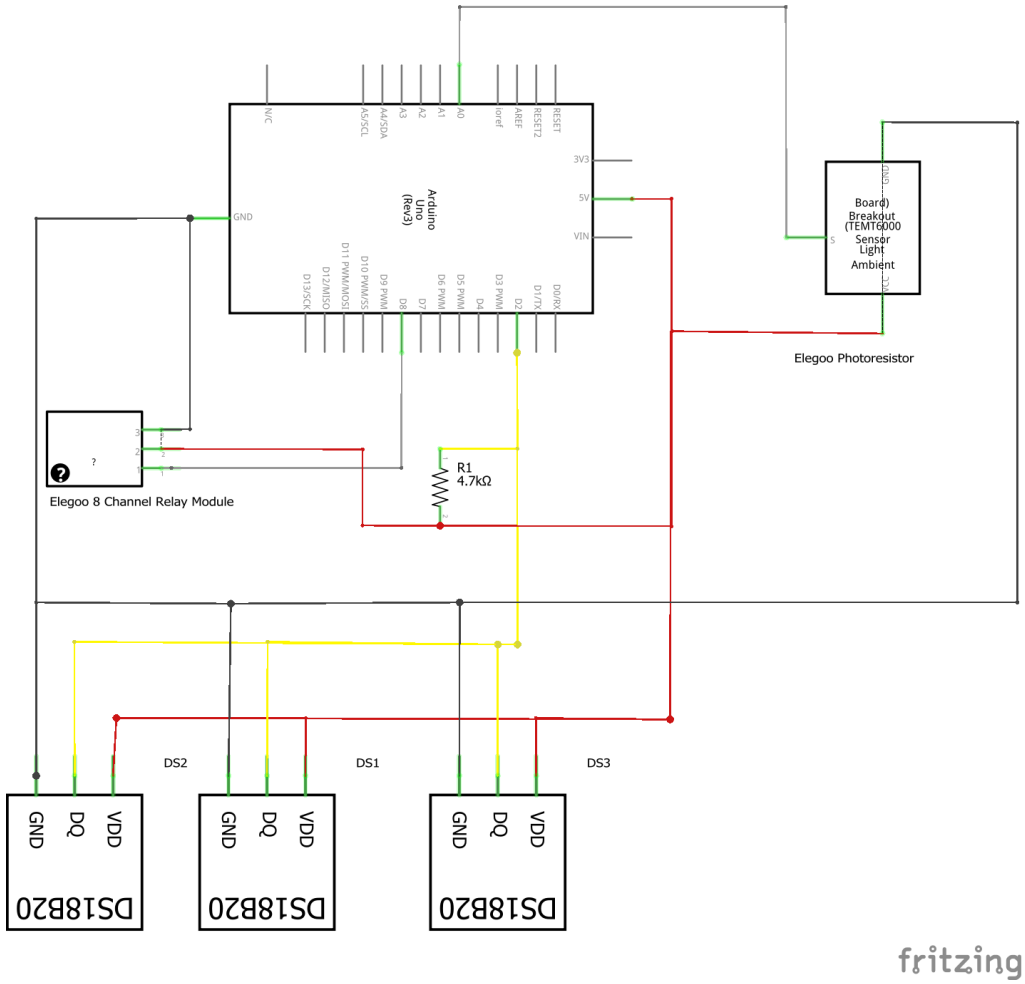


Figure 8 : Les branchements sur la vue schématique

Versions

Les logiciels et les librairies sont en constante évolution, de ce fait le tuto est valable aujourd'hui, même peut-être que demain il sera devenu moins précis à cause des mises à jour.

Voici donc les versions sur lesquelles est basé ce tutorial :

- Homeassistant :
  - Host : Home Assistant OS 7.6
  - Supervisor : 2022.03.5
  - Core : 2022.4.1
- Mosquitto broker : 6.0.1
- File editor : 5.3.3
- Librairies Arduino :
  - SPI.h : Pas de N° de version trouvée !!
  - Ethernet.h : 2.0.0
  - PubSubClient.h : 2.8.0
  - OneWire.h : 2.3.6
  - DallasTemperature.h : 3.9.0
  - avr/wdt.h : Pas de N° de version trouvée !!

Installation côté Homeassistant

J'avais un vieux Raspberry qui trainait dans un carton du garage que j'ai dépoussiéré en rangeant. Du coup, comme je vais bientôt me lancer dans un projet d'installation domotique dans les prochains mois, je suis en train de réfléchir à comment je peux faire au mieux, pour optimiser entre coût et fonctionnalités.

J'ai déjà installé une domotique avec le couple Serveur WES et Jeedom, mais je voulais changer Jeedom (j'ai eu beaucoup de soucis de choses qui ne fonctionnaient plus à chaque grosse mise à jour). Et de plus beaucoup de modules complémentaires sont payants, alors que sous Homeassistant, ils sont tous gratuits !

Comme expliqué, j'ai donc utilisé mon vieux Raspberry 3B+, pour installer homeassistant.

Je ne vais pas m'étendre sur l'installation, c'est hyper simple et rapide en flashant le fichier.
Par contre, j'ai testé la solution avec une carte SD et au bout de quelques jours elle m'a lâché (trop de cycles d'écritures) !
J'ai donc acheté un disque SSD de 250Go que j'ai branché en USB sur le RPI sur lequel j'ai installé HomeAssistant.
Vous trouverez plein de tutoriels pour modifier le fichier de configuration et avoir la possibilité de booter sur une clef ou un disque dur USB sur RPI3 (ou 4).

Une fois l'installation de Homeassistant derrière vous et lors de votre première connexion sur l'interface, il faudrait vous familiariser avec la manipulation des fichiers de configuration. Car malheureusement certaines actions ne sont pas encore réalisables via l'interface dédiée aux scénarios, etc…

Mais pas de panique, vous n'aurez pas besoin forcément de passer du temps à comprendre comment cela fonctionne (même si c'est évidemment conseillé), je vous donne toutes les clefs dans ce document.

Donc, on va installer les modules additionnels qui vont bien.

Module additionnel File editor

Vous retrouverez l'excellent tutoriel sur le site de la communauté FR d'Homeassistant :

https://forum.hacf.fr/t/installer-file-editor-et-modifier-vos-fichiers-de-configuration-depuis-ha/203

Ce module est important, car il vous permet de modifier les fichiers de configuration d'Homeassistant directement depuis l'interface. Autrement il faut se connecter au RPI (en direct ou via SSH), mais ça complique les choses, alors autant rester simple 😊 !

N'oubliez pas de cocher l'option « show in sidebar », cela vous permettra de voir apparaitre « File editor » dans le menu de gauche. Ça en facilitera l'accès.

Module additionnel Mosquitto broker

Je ne vais pas rentrer dans les détails de ce qu'est un broker, vous trouverez pleins de documents l'expliquant mieux que moi. Sachez simplement qu'un broker MQTT, tel que Mosquitto, est un serveur de communication pour le protocole MQTT. Sur notre réseau local sur lequel sera installé l'arduino et Homeassistant, il faut donc un serveur MQTT pour relayer les messages entre les utilisateurs (dans l'exemple de ce tuto, 2 utilisateurs : homeassistant et l'Arduino).

Avant tout, vérifions que vous avez activé le mode « avancé » sous Hoemassistant :

Cliquez sur votre nom d'utilisateur en bas à gauche et activer « mode avancé » :

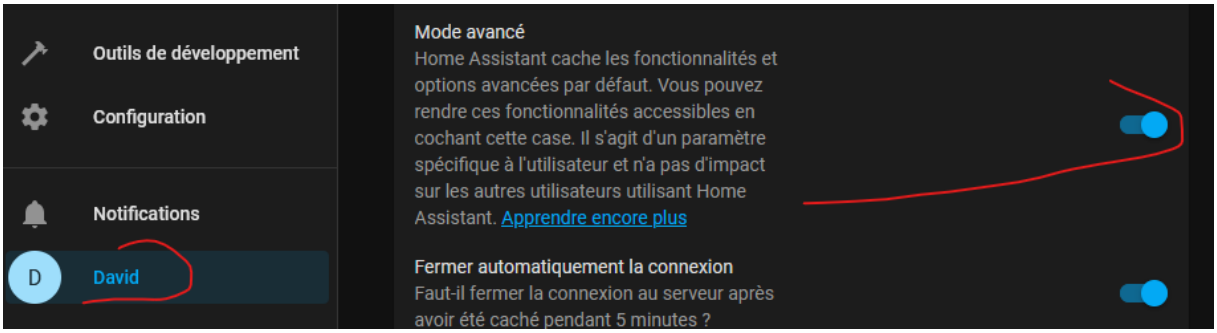
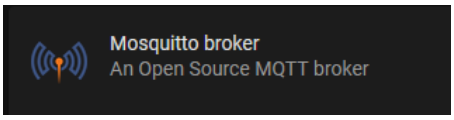


Figure 9 : Activation du mode avancé

Puis cliquons sur la gauche sur « configuration » et « Modules complémentaires, Sauvegardes et Superviseur » pour ouvrir la fenêtre d'ajout de nouveaux modules complémentaires.

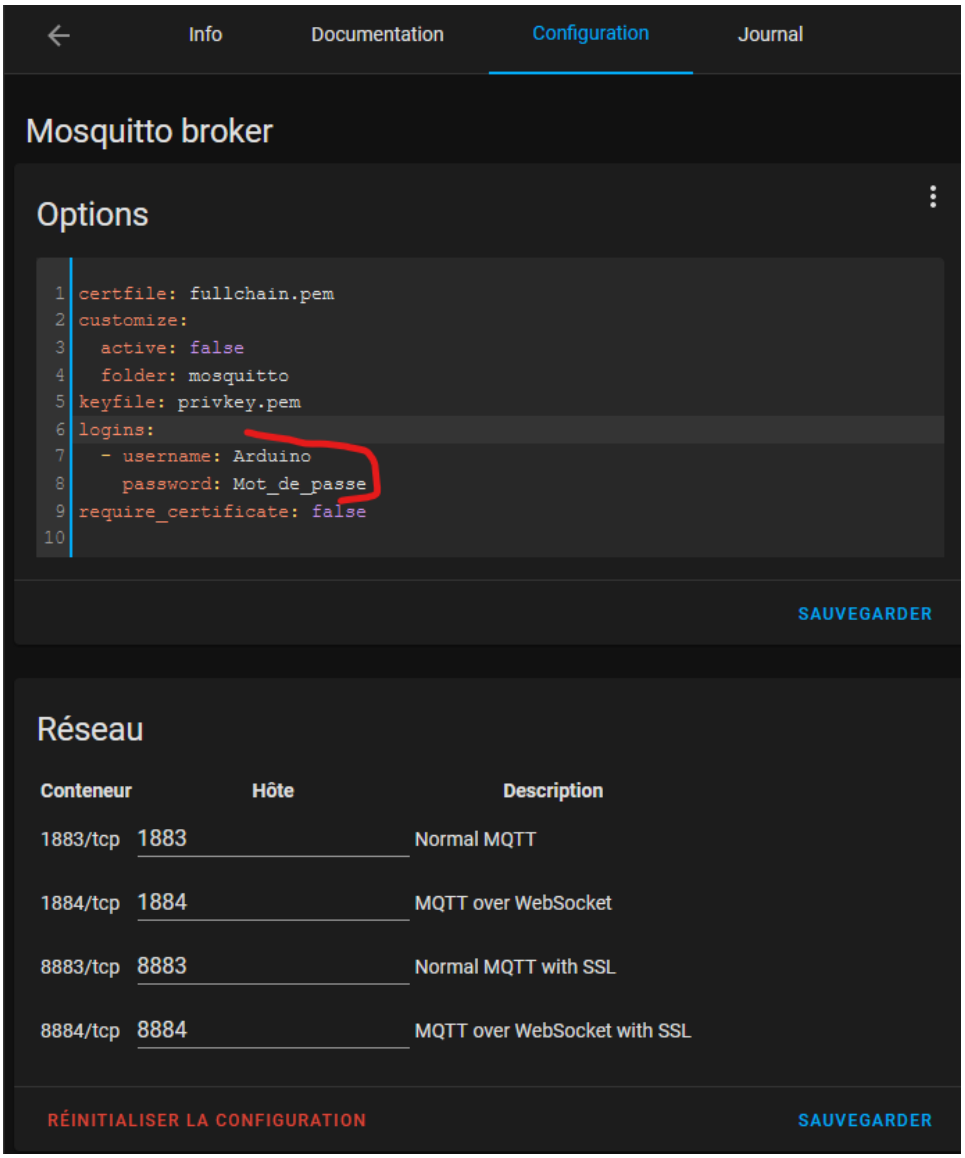
Cliquez sur le bouton « Boutique des modules complémentaires » et cherchez puis installer le module « Mosquitto broker » :



Une fois installé, il devrait s'afficher dans la fenêtre des modules complémentaires, mais chez moi ça n'a pas fonctionné, j'ai donc redémarré mon RPI, en allant sur « configuration » , « Modules complémentaires, Sauvegardes et Superviseur » puis en cliquant sur l'onglet tout à droite en haut, nommé « Système ». Puis choisir « redémarrer l'hôte ».

Une fois le redémarrage terminé, revenir sur la fenêtre des modules complémentaires, vous devriez trouver Mosquitto Broker. On clique dessus. On clique sur l'onglet « Configuration » en haut de la page.

On va entrer l'identifiant et le mot de passe à la connexion au serveur, comme sur la copie d'écran ci-dessous, choisissez un couple identifiant/mot de passe.
Dans notre exemple l'identifiant est **Arduino** et le mot de passe est **Mot_de_passe**.



On ne touche pas aux réglages des ports et on sauvegarde le tout.

Puis ensuite on retourne sur l'onglet « Info » de notre module additionnel (Figure 10) et on va s'assurer que :

1. « Lancer au démarrage » est actif
2. « Chien de garde » est actif

L'option chien de garde (watchdog en anglais) est utile pour s'assurer qu'en cas de crash de Mosquitto broker pour quelque raison que ce soit, il redémarrera tout seul comme un grand (sans devoir redémarrer le Raspberry ou Homeassistant).

Puis on clique sur démarrer en bas à droite.

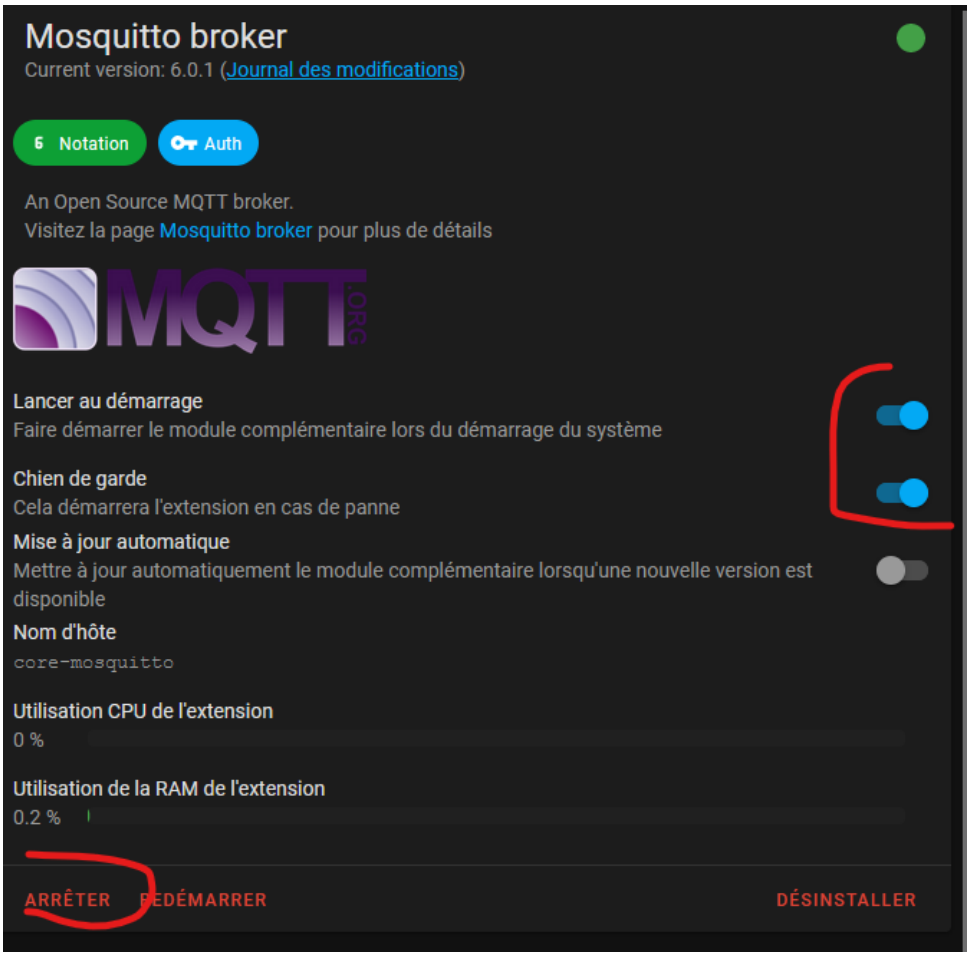


Figure 10 : Une fois le démarrage terminé, voilà à quoi devrait ressembler votre écran

Fichiers de configuration Homeassistant et création des entités

Création des entités/sensors, c'est comme vous voulez ! Bref on va créer les « items » qui vont nous permettre d'avoir ces entités présentes dans homeassistant.

Comme cette intégration est manuelle, il faut passer par une modification manuelle du fichier de configuration appelé « configuration.yaml ».

On va donc cliquer sur « File editor » sur le menu de gauche. Si vous ne tombez pas sur le fichier « configuration.yaml », il suffit de cliquer sur l'icône du dossier  en haut à gauche de la fenêtre et sélectionner le bon fichier (Figure 11).

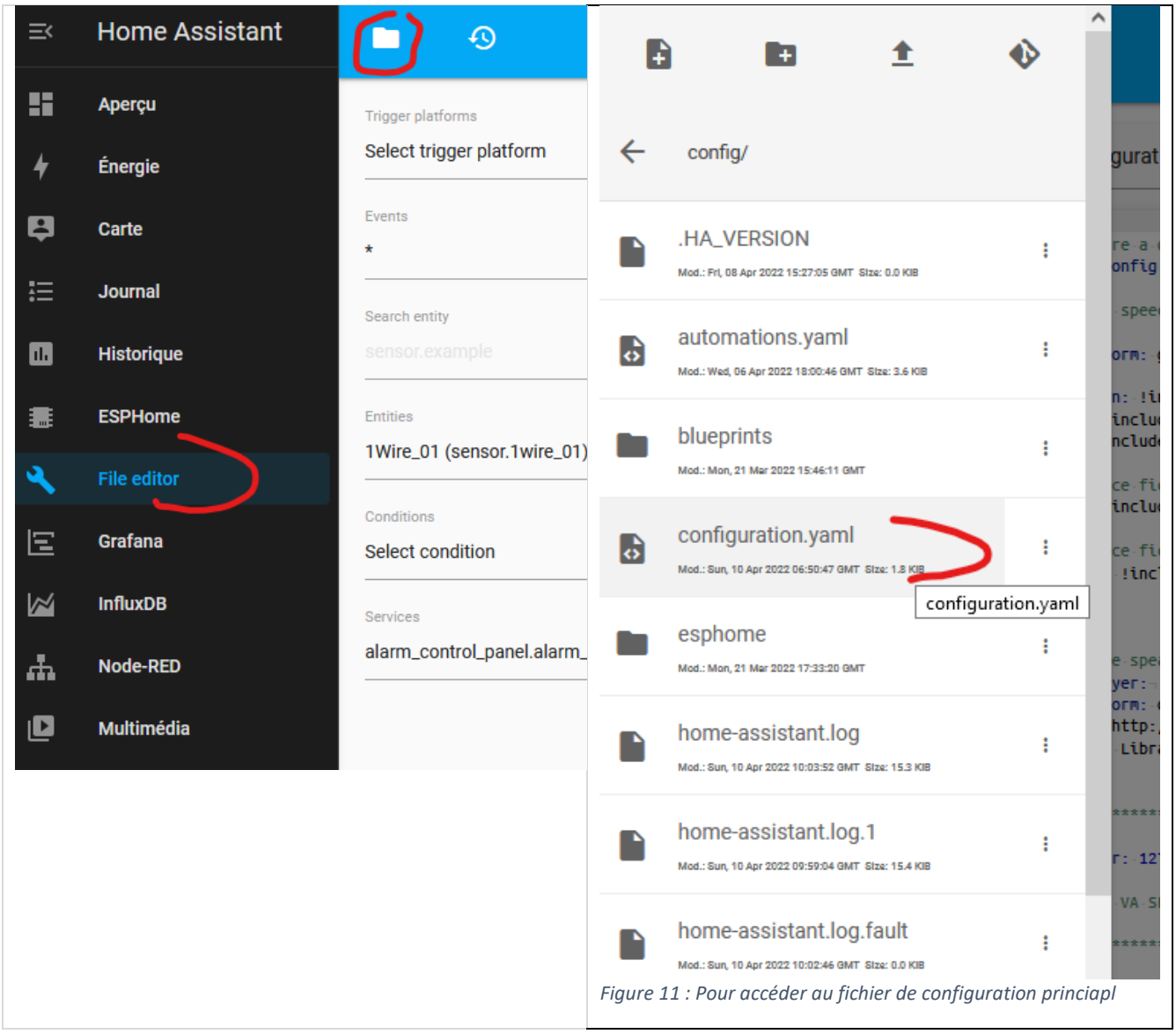


Figure 11 : Pour accéder au fichier de configuration princiapl

Voici la partie de mon fichier de configuration « configuration.yaml » dans laquelle je retrouve la configuration de mes entités rentrées manuellement dans Homeassistant :

```
# ***************  DEBUT DE L'AJOUT CONFIGURATION MQTT   ******************
mqtt:
    broker: 127.0.0.1 #Ici c'est le localhost puisque le broker est installé sur
le RPI où Homeassistant est installé.

# ********************* Sensors MQTT ************************
sensor:
# Ce sensor a été créé au début pour les tests de communication depuis le arduino
vers HA
  - platform: mqtt
    name: "Humeur de David"
    state_topic: "home-assistant/david/humeur"


# Ce sensor a été créé pour afficher la mesure de luminosité de la sonde
photorésistante
  - platform: mqtt
    name: "Brightness"
    state_topic: "home-assistant/sensor01/brightness"
```

```
    unit_of_measurement: "cd"




# A partir de là, je vais créer mes 10 sondes de températures 1WIRE 18DS20B
  - platform: mqtt
    name: "1Wire_01"
    state_topic: "home-assistant/sensor01/1Wire_01"
    unit_of_measurement: "°C"
    #Ici je divise par 100 et arrondi à 2 chiffres après la virgule, car sur
l'Arduino, j'ai multiplié la valeur par 100 (le protocole MQTT n'accepte de
transférer que des entiers apparemment !)
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_02"
    state_topic: "home-assistant/sensor01/1Wire_02"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_03"
    state_topic: "home-assistant/sensor01/1Wire_03"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_04"
    state_topic: "home-assistant/sensor01/1Wire_04"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_05"
    state_topic: "home-assistant/sensor01/1Wire_05"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_06"
    state_topic: "home-assistant/sensor01/1Wire_06"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_07"
    state_topic: "home-assistant/sensor01/1Wire_07"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_08"
    state_topic: "home-assistant/sensor01/1Wire_08"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"

  - platform: mqtt
    name: "1Wire_09"
    state_topic: "home-assistant/sensor01/1Wire_09"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"
```

```
  - platform: mqtt
    name: "1Wire_10"
    state_topic: "home-assistant/sensor01/1Wire_10"
    unit_of_measurement: "°C"
    value_template: "{{ value | multiply(0.01) | round(2) }}"




#Définition d'un switch MQTT (qui me permettra d'activer le relais connecter à
l'arduino depuis homeassistant
switch:
  - platform: mqtt
    name: "Virtual_Switch_01"
    state_topic: "home-assistant/switch/Virtual_Switch_01/HA_Read" #Nom du canal
pour lire la valeur actuelle
    command_topic: "home-assistant/switch/Virtual_Switch_01/HA_Write" # Nom du
canal pour publier une valeur
    qos: 2 #2 = message envoyé qu'une seule fois
    payload_on: 0 # or "on", depending on your MQTT device
    payload_off: 1 # or "off", depending on your MQTT device
    retain: false # or true -> SI on met à true le message reste en "mémoire"
jusqu'au prochain accès au broker - Je n'ai pas trouvé de soucis avec le
fonctionnement
    #en QOS 2 sans le retain en utilisant mon shield arduino Etherent, puet-être
qu'avec une mauvais signal wifi cela pourrait être nécessaire de modifier ces
paramètres ?

# ***************************************************************************
```

Avant de démarrer les explications, il faut savoir que Homeassistant est très regardant sur ce que j'appelle la « hiérarchisation par tabulation ».

Les sections principales sont (mqtt: / switch: ou encore sensor:) doivent être placés sans espace ou tabulation sur leur gauche.

Puis « - platform » est déplacée à droite d'UNE tabulation et tout le reste de DEUX tabulations.

Sans respecter cette règle vous aurez des erreurs lors de l'enregistrement des fichiers de configuration. C'est un peu « chiant » au début, mais ça s'avère être une très bonne idée pour améliorer la lisibilité du code. Pour les bordéliques comme moi, ça peut aider !

Autre point, l'ajout d'un # devant un texte, commente ce même texte. Un conseil, commentez à fond vos codes surtout si vous n'y mettez pas le nez souvent, autrement il vous faudra de nouveau vous casser la tête à comprendre encore une fois ce que vous avez fait après plusieurs semaines (ou mois pour les plus chanceux) que vous ne vous rappellerez plus de comment vous avez fait cette P####n de m###e !!!

Allons-y pour l'explication de chaque partie :

```
mqtt:
    broker: 127.0.0.1
```

C'est la définition de l'adresse du broker (serveur) MQTT.
Ce serveur est installé chez moi, sur mon Raspberry (installation réalisée via Homeassistant et le

module additionnel idoine). Comme Homeassistant est également installé sur mon Raspeberry on peut laisser l'adresse IP 127.0.0.1, qui correspond à l'adresse de la même machine.

Si vous avez installé votre broker sur une autre machine, il faudra modifier cette IP pour la bonne.

```
sensor:
# Ce sensor a été créé pour afficher la mesure de luminosité de la sonde
photorésistante
  - platform: mqtt
    name: "Brightness"
    state_topic: "home-assistant/sensor01/brightness"
    unit_of_measurement: "cd"
```

Au début on commence avec la section « sensor », ce qui veut dire que toutes les entrés suivantes seront de type « sensor », jusqu'à ce qu'on arrive à la prochaine section.

Comme le commentaire l'indique, cette première entité a été créée pour récupérer la valeur de luminosité qui sera envoyée par la sonde photorésistive branchée sur l'arduino.

On y voit qu'elle est connectée au serveur MQTT (- platform : mqtt).
Son nom (ce qui sera affiché sur l'interface de Homeassistant).
Et son « state_topic », qui est le nom du canal ou plutôt le nom du message (dans lequel on écrira la valeur de luminosité).
Le « unit_of_measurement » pour les anglophobes correspond à l'unité de mesure (ici candela pour cd). C'est donc l'unité qui sera utilisée par Homeassistant pour l'affichage sur l'interface.

```
  - platform: mqtt
    name: "1Wire_01"
    state_topic: "home-assistant/sensor01/1Wire_01"
    unit_of_measurement: "°C"
    #Ici je divise par 100 et arrondi à 2 chiffres après la virgule, car sur
l'Arduino, j'ai multiplié la valeur par 100 (le protocole MQTT n'accepte de
transférer que des entiers apparemment !)
    value_template: "{{ value | multiply(0.01) | round(2) }}"
```

Ici on va déclarer les sondes de températures 1Wire.
Je ne reviens pas sur ce que j'ai déjà expliqué ci-dessus (c'est identique).
La « value_template » est un calcul réalisé sur ce que reçoit homeassistant. Je ne suis pas arrivé à envoyer un chiffre à virgule pour la température (c'est balo quand même !). Du coup, j'ai multiplié par 100 sur mon arduino pour avoir un chiffre entier (22.31°C, devient donc 2231) et donc je dois faire l'opération inverse (diviser par 100) quand homeassistant réceptionne la valeur. Le round (2) dit simplement que je veux un chiffre avec 2 décimales (2 chiffres après la virgule pour ceux qui n'écoutaient pas en cours de Maths).

Bon pour l'instant c'est cool, mais toutes les entités qu'on vient de créer ne servent qu'à envoyer du contenu du Arduino vers homeassistant. Maintenant ce que je voudrais bien, c'est piloter un relais depuis homeassistant pour rendre ça un peu plus fun !

```
#Définition d'un switch MQTT (qui me permettra d'activer le relais connecter à
l'arduino depuis homeassistant
switch:
  - platform: mqtt
    name: "Virtual_Switch_01"
    state_topic: "home-assistant/switch/Virtual_Switch_01/HA_Read" #Nom du canal
pour lire la valeur actuelle
```

```
    command_topic: "home-assistant/switch/Virtual_Switch_01/HA_Write" # Nom du
canal pour publier une valeur
    qos: 2 #2 = message envoyé qu'une seule fois
    payload_on: 0 # or "on", depending on your MQTT device
    payload_off: 1 # or "off", depending on your MQTT device
    retain: false # or true -> SI on met à true le message reste en "mémoire"
jusqu'au prochain accès au broker - Je n'ai pas trouvé de soucis avec le
fonctionnement
    #en QOS 2 sans le retain en utilisant mon shield arduino Etherent, puet-être
qu'avec une mauvais signal wifi cela pourrait être nécessaire de modifier ces
paramètres ?
```

Alors ici on voit une variante sur les « topic » :

- State_topic -> Nom du canal ou du message dont homeassistant aura besoin, comme pour les autres entités plus haut, pour lire une valeur depuis Arduino vers Homeassistant
- Command_topic -> Tada… C'est le nom du canal (ou du message) qu'homeassistant utilisera pour envoyer des infos/commandes, bref tout ce que vous voulez vers Arduino

Pour le reste :

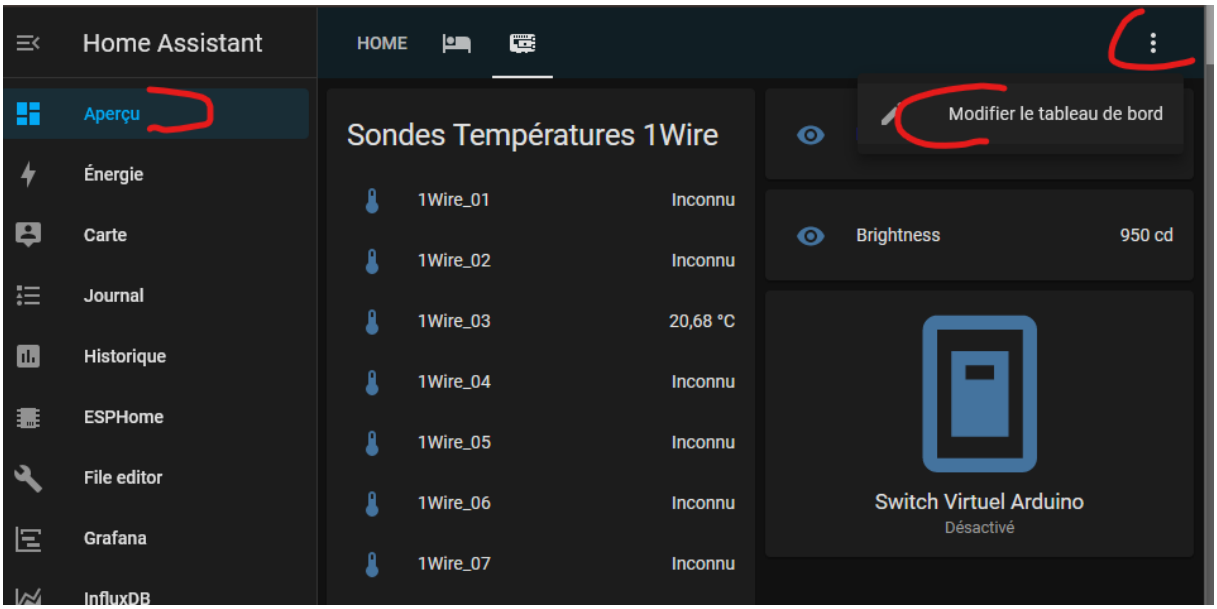
- Qos -> cette valeur définit la redondance (qos = quality of service) des messages envoyés (en mode 2 comme chez moi, le message avec sa valeur n'est envoyé qu'une seule fois). Ça fonctionne bien sur une liaison ethernet, maintenant si vous avez un wifi avec un signal faible, peut-être qu'il faut passer à 1 ou 0 (je vous laisse regarder en détail à quoi correspond ces valeurs : tapez « mqtt qos » dans votre moteur de recherche préféré)
- Payload_on -> c'est la valeur du message envoyée, donc ici quand mon switch est actif sur homeassistant (quand vous cliquez dessus et que vous le faites passer en mode « allumé »), le contenu du message (ou du canal) sera de « 0 ».
- Payload_off -> inverse de payload_on !
- Retain -> En mettant sur True le message reste en « mémoire » et n'est pas écrasé par un autre message du même canal. J'ai pas trop creusé cette partie, car cela fonctionne très bien en laissant la valeur à false. C'est peut-être intéressant pour ceux qui ont une connexion sans fil pas très fiable.

Il est grand temps de faire une sauvegarde, il serait dommage de tout perdre maintenant (même si vous n'avez fait que des copiers-collers 😉, pour ça : « Configuration » – « modules complémentaires, sauvegardes, superviseur » - onglet « sauvegardes » - « Créer une sauvegarde ». On patiente quelques minutes.

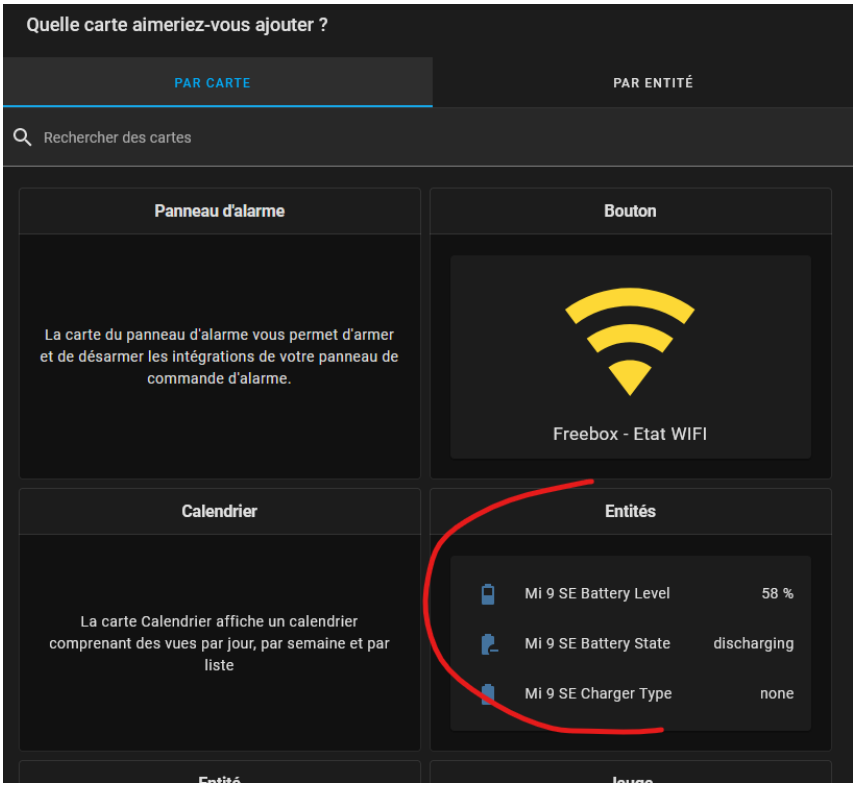
Je vous conseille de redémarrer Homeassistant, pour que les entités soient créées au démarrage suivant du système, pour cela : « Configuration » – « modules complémentaires, sauvegardes, superviseur » - onglet « système » - « redémarrer l'hôte ». On patiente quelques minutes.

Maintenant on va ajouter les cartes correspondantes à nos entités sur l'interface graphique.

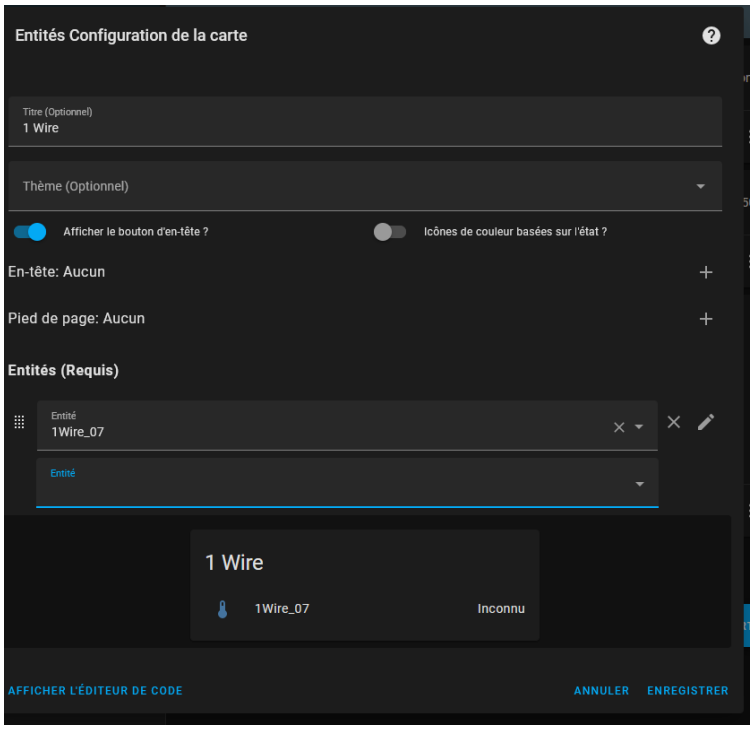
Cliquez sur le menu de gauche « Aperçu », puis sur les 3 points verticaux en haut à droite de la fenêtre, puis sur « modifier le tableau de bord »



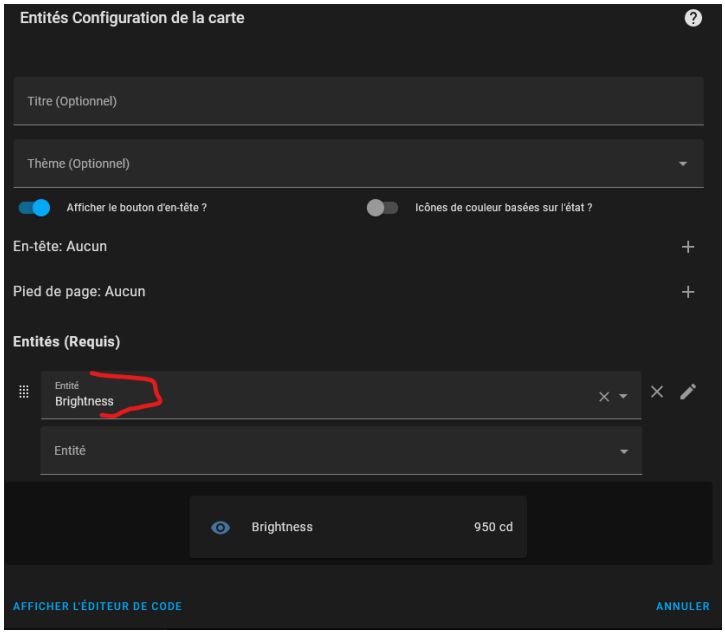
Puis sur le bouton « ajouter une carte »



Puis sur « entités » pour commencer.

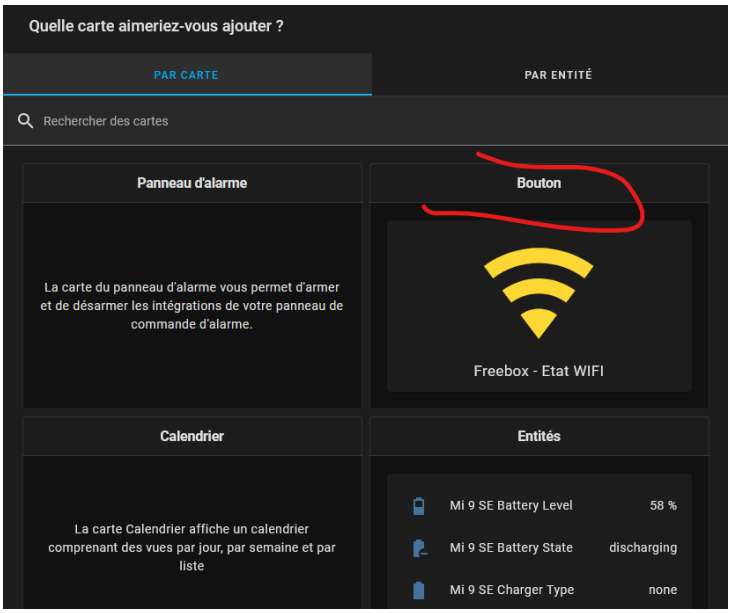


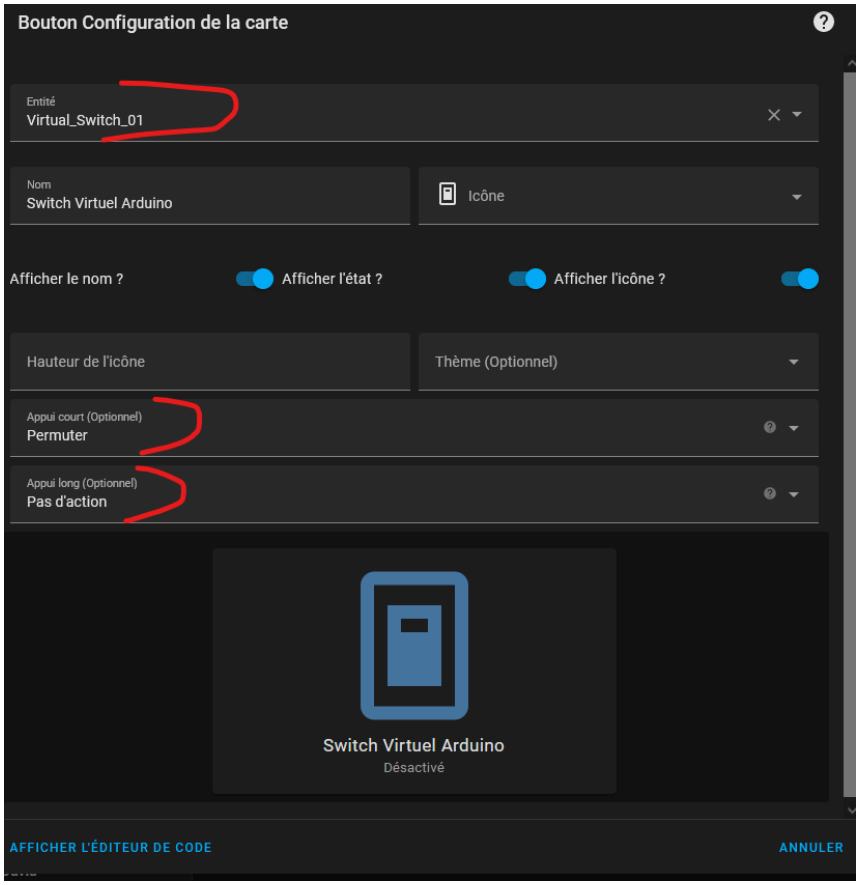
Ici dans la liste déroulante « entités » vous pouvez chercher vos 10 sondes de température 1Wire, puis cliquer sur « Enregistrer ».



On va créer une nouvelle carte « entité » et choisir « brightness ».

Puis une dernière de type « bouton » :





Le remplir avec le nom du switch virtuel et mettre les options telles quelles.

Voilà, on vient de faire le tour sur Homeassistant pour l'usage des modules qu'on utilisera dans ce projet de tuto.

Avant de passer à l'arduino, je vous conseille également d'installer les modules suivants :

- InfluxDB (base de données pour sauvegarder les enregistrements sur le long terme)
- Grafana (pour afficher vos infos stockées dans influxDB dans de jolis graphiques)
- SSH & Web Terminal (pour pouvoir accéder via SSH au RPI pour pouvoir faire certaines très rares opérations à distance)

Je vous laisse vous débrouiller avec les tutos sur ces sujets qui sont bien faits sur le site de la communauté française de homeassistant : https://forum.hacf.fr/

Installation côté Arduino

Je vous laisse brancher le matos conformément au schéma.

Notez que le shield Ethernet que j'utilise, me permet d'avoir une connexion filaire avec un câble RJ45. Je n'utilise donc pas de wifi pour le moment pour gérer mon matos et les communications (je suis un fervent défenseur du filaire 😉, en tout cas quand c'est possible sans trop alourdir la facture).

Voici le code complet pour l'arduino :

```
/*
 * Ce programme nécessite :
 *  - une photo résistance branchée (ou tout capteur qui sort du signal
analogique) sur le pin A0
 *  - une carte relais branchée sur le PIN D8
 *  - Un Arudino UNO
 *  - Un shield Ethernet (utilisé le ks0156 de keyestudio (basé sur un W5100
ethernet shield)
 */

//Nous allons commencer notre code en incluant les bibliothèques nécessaires.
//La première est OneWire.h, qui permet d'interagir avec les appareils utilisant
le protocole OneWire.
//La seconde est DallasTemperature.h, qui exposera quelques méthodes de plus haut
niveau qui nous permettront d'interagir avec le capteur DS18B20.
#include <OneWire.h> // Utilisé pdt les tests la version 2.3.6
#include <DallasTemperature.h> // Utilisé pdt les tests la version 3.9.0


//****************Les librairies pour la connexion internet et la MQTT
#include <SPI.h> // Version utilisée : Pas de N° de version trouvée !!
#include <Ethernet.h> // Version utilisée : 2.0.0
#include <PubSubClient.h> // Version utilisée : 2.8.0

////***************** Librairie pour le wtachdog//////////
#include <avr/wdt.h> // Version utilisée : Pas de N° de version trouvée !!
//*******************************


////////////CALCUL DE LA TAILLE DU TABLEAU Sondes_1Wire Partie #1 ///////////////
#define ROW_COUNT(array)     (sizeof(array) / sizeof(*array))
#define COLUMN_COUNT(array) (sizeof(array) / (sizeof(**array) *
ROW_COUNT(array)))
////////////////////////////////////////////////////////////////////////////////


// COnfiguration de l'adresse MAC pour récupérer l'adresse IP on va utiliser le
code  "if (Ethernet.begin(mac) == 0) {" plus bas
uint8_t mac[6] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x06};
```

```
//Ici c'est une variable qui permettra d'enregistrer la valeur de la luminosité
char buf[4]; // Buffer to store the sensor value

// Définition de l'adresse du broket MQTT (configuration d'un "bail statique" sur
la box !) : Freebox -> Paramètres de la freebox -> DHCP
const char* MQTTserver = "192.168.1.50";


// Initialisation des objets Ethernet & MQTT
EthernetClient ethClient;
PubSubClient mqttClient(ethClient);

//Après les includes, nous aurons besoin d'un objet de la classe OneWire, qui
permet d'échanger des octets avec des appareils connectés au bus OneWire.
//Cet objet sera utilisé en interne par la bibliothèque DallasTemperature pour
échanger les données avec le capteur.
//En entrée, le constructeur de cette classe reçoit le numéro du pin du
microcontrôleur connecté au(x) capteur(s).
//#define ONE_WIRE_BUS 2
// Ici pas besoin de délcarer un autre bus, on peut brancher tous les capteurs
sur la même entrée du Arduino
// Setup a oneWire instance to communicate with any OneWire devices (not just
Maxim/Dallas temperature ICs)
OneWire oneWire(2);



// Pass our oneWire reference to Dallas Temperature.
//Nous aurons également besoin d'un objet de la classe DallasTemperature, qui est
celui que nous utiliserons dans notre code pour obtenir l'adresse du capteur.
//Comme entrée du constructeur de cette classe, nous passerons l'adresse de notre
objet OneWire précédemment instancié.
DallasTemperature Temp_sensors(&oneWire);

//////////////////////DECLARATION DES CONSTANTES 1WIRE/////////////////////////
//Ici j'ai récupéré les numéros hexadecimals de chaque sonde et je les affecte à
mes constantes
//chaque chiffre héxadécimal doit prendre un "0x" devant le numéro héxa

//Déclaration d'un tableau de 10 entrées qui reprends les devices adresse de
chaque sonde (le numéro hexadecimal de chaque)
DeviceAddress Sondes_1Wire[] = {
  { 0x28, 0x20, 0xEA, 0xEA, 0x17, 0x20, 0x6 , 0xB1 },    //Sonde 01
  { 0x28, 0xE1, 0xB6, 0xF1, 0x17, 0x20, 0x6 , 0x1 },     //Sonde 02
  { 0x28, 0x99, 0x12, 0x92, 0x12, 0x21, 0x1 , 0x7E },     //Sonde 03
  { 0x28, 0x24, 0x22, 0x1F, 0x18, 0x20, 0x6 , 0xDE },    //Sonde 04
  { 0x28, 0x19, 0xDA, 0x39, 0x12, 0x21, 0x1 , 0xB5 },    //Sonde 05
  { 0x28, 0x5A, 0x6D, 0x1C, 0x18, 0x20, 0x6 , 0x5B },    //Sonde 06
  { 0x28, 0x69, 0x54, 0xF, 0x18, 0x20, 0x6 , 0x28 },     //Sonde 07
  { 0x28, 0x1B, 0x73, 0xDF, 0x17, 0x20, 0x6 , 0x29 },    //Sonde 08
  { 0x28, 0x36, 0x53, 0xE6, 0x17, 0x20, 0x6 , 0xF6 },    //Sonde 09
  { 0x28, 0x63, 0xBA, 0x8, 0x18, 0x20, 0x6 , 0x19 }      //Sonde 10
};



int nbre_Sensors;


/*
 * Ici c'est la fonction qui permet de traiter les messages qui arrivent depuis
le broker MQTT (et donc depuis HA)
 *  l'arguement "topic" -> c'est tout simplement le nom de canal par lequel
transite le message
 *  (par exemple "home-assistant/switch/Virtual_Switch_01/HA_Read")
 *  l'argument "payload" -> c'est le contenu du message
 *  "length" -> c'est la longueur du message
 *
```

```
 *  Cette fonction est automatiquement appelée lorsque le broquer MQTT envoie un
message à l'arduino (ou plutôt
 *  lorsque l'arduino lit un message)
 */




void callback(char* topic, byte* payload, unsigned int length)
{
    // handle received message

String Str;

  Serial.print(F("**** Message arrivé depuis MQTT broker **** -> [ "));
  //Imprime le chemin sur le broker
  Serial.print(topic);
  Serial.print(F(" ] -> ' "));
  for (int i=0;i<length;i++) {//ici le lenght c'est la variable de la fonction
callback
    Serial.print((char)payload[i]);
    Str = Str + (char)payload[i];
  }
  Serial.println(F(" '"));



//************ Fonction pour le switch virtuel 01******************

/*
 * Nom du canal qui fonctionne en relation avec le switch
 * J'ai testé avec if(topic == "home-assistant/switch/Virtual_Switch_01/HA_Write"
&& Str == "1"){
 * ou if(topic == canal_switch_virtuel && Str == "1"){
 * mais cela ne fonctionne pas !
 *
 * Seul moyen c'est de comparer les 2 valeurs en amont avec strcomp (renvoie 0 si
les 2 chaines de caractères sont
 * strictement indentiques !
 */
char* Canal_Switch_Virtuel = "home-assistant/switch/Virtual_Switch_01/HA_Write";
int valeur_Canal = strcmp(topic, Canal_Switch_Virtuel);

/*
 * Ici ma logique IF va contrôler que le message est bien adressé au switch
virtuel 01 (utile si plusieurs switchs
 * sont exitsants et à tester) et en fonction de la valeur qui arrive depuis
Homeassistant on va activer ou non le
 * relais
 * Attention, le delay de 5seconde dans mon "void loop()" vient ralentir la
réactivité du système en bloquant
 * tout l'arduino, regarder pour trouver une solution alternative avec une
teporisation autre que le delay !
 */
  if(valeur_Canal == 0 && Str == "1"){
    Serial.println(F("Bouton appuyé, envoie d'un 1 -> Bouton INACTIF sous
HomeAssistant"));
    mqttClient.publish("home-assistant/switch/Virtual_Switch_01/HA_Read", "1");
    digitalWrite(8,HIGH);
    Serial.println(F("Relais ouvert"));

  }
  else if (valeur_Canal == 0  && Str == "0"){
    Serial.println(F("Bouton appuyé, envoie d'un 0 -> Bouton ACTIF sous
HomeAssistant"));
    mqttClient.publish("home-assistant/switch/Virtual_Switch_01/HA_Read", "0");
    digitalWrite(8,LOW);
```

```
    Serial.println(F("Relais fermé"));
  }

  //*************************************

}



void setup() {
  // Useful for debugging purposes
  //pas vu de différences de latence/réactivité entre 115200 et 9600bauds !
  Serial.begin(115200);


/* Le Serail.println(F("Ma chaine de caractères")); permet de suver la mémoire
SRAM en déplaçant
 *  les chaines de caractères constantes depuis la SRAM vers la mémoire Flash.
Habituellement la mémoire FLASH
 *  est plus disponible que la SRAM.
 */

  Serial.println(F("Démo MQTT Arduino (entrée + sortie avec Relais piloté)"));
  Serial.println();

//Configuration d'une carte relais, qui viendra commuter selon le retour d'un
signal MQTT depuis HomeAssistant
  Serial.println(F("Sortie Relais branchée sur le Digital 8"));
  pinMode(8,OUTPUT);
  //Désactivation du relais (chez moi le relais de la carte (Elegoo - 8 channel
5V Relay module) est inversé
  digitalWrite(8,HIGH);


//watchdog timer with 8 Seconds time out
//Ce watchdog permet après 8s non ressté (arduino freezzé) de redémarre le CPU !
   wdt_enable(WDTO_8S);


/* Configuration de l'adresse IP de l'arduino en faisant appel au serveur DHCP
sur le réseau local
 *  (= de la freebox)
 *
 */
  if (Ethernet.begin(mac) == 0) {
    Serial.println(F("Impossible de configurer Ethernet via serveur DHCP !"));
    for (;;);
  }


  Serial.println(F("Ethernet configuré via DHCP"));
  Serial.print(F("IP address: "));
  Serial.println(Ethernet.localIP());
  Serial.println(F("------------"));

  // Set the MQTT server to the server stated above ^
/*  Configuration du broker MQTT (serveur MQTT), en lui indiquant l'adresse IP
notée en constante plus haut
*   et le port (1883 par défaut)
*/
  mqttClient.setServer(MQTTserver, 1883);
  Serial.println(F("Client MQTT configuré !"));

 Serial.println(F("Tentative de connexion au serveur avec l'ID 'Arduino'"));
  // Attempt to connect to the server with the ID "myClientID"
/*Ici on essaye de se connecter au broker MQTT, donc ici il faut rentrer :
```

```
*   connect(ID, user, pwd) dont user et pwd sont respectivement les noms
utilsateur et mot de passe définis
*   dans la configuration du module complémentaire Mosquitto broker.
*   Je ne sais pas à quoi sert le "ID" ?
  */
  if (mqttClient.connect("Arduino","Arduino","Mot_de_passe"))
  {
    Serial.println(F("La connection a été établie, well done !"));
   }
  else
  {
    Serial.println(F("Il semblerait que le serveur ne réponde pas..."));
  }


/* 1er appel au broker MQTT
 *  La commande Publish, est uniquement nécessaire pour envoyer au broker du
contenu.
 *  Si on veut écouter depuis le broker, il faudra utiliser setCallback et
s'inscrire à un sujet
 *  sur le broker (voir plus bas mqttClient.setcallback & mqttClient.subscribe)
 *
 *  Le fonctionnement est le suivant : mqttClient.publish("home-
assistant/switch/Virtual_Switch_01/HA_Read", "1");
 *  "home-assistant" -> Définit évidemment que c'est HhomeAssastant qui est
l'entité
 *  "switch" -> c'est le type de "capteur" utilisé sous HA (ici un switch, créé
dans configuration.yaml)
 *  "Virtual_Switch_01" -> c'est ce qui correspond au "name" du même "capteur",
créé dans configuration.yaml)
 *  "HA_Read" -> Correspond au nom du cannal MQTT sur lequel Homeassistant
écoutera (et donc c'est via ce cannal
 *  que l'arduino enverra les commandes/ordres à HA. Cela correspond à la ligne
"state_topic" du configuration.yaml
 *
 *  Dans cet exemple, l'arduino, "dit" à Homeassistant de passer le
switch_virtuel_01 à 1.
 *
 * Ici j'insére cet envoi avant le callback, comme ça on configure les boutons
sous HA,
 * sans que cela ne fasse une intéraction avec l'arduino. Sans initialiser le
status du bouton sous HA est "inconnu".
 */
mqttClient.publish("home-assistant/switch/Virtual_Switch_01/HA_Read", "1");
/*Initialisation de la procédure permettant d'écouter le broker MQTT (il permet à
l'Arduino de lire les messages
 * envoyés par Homeassistant
 */
mqttClient.setCallback(callback);


Serial.println();
  Serial.println(F("Début de configuration des sondes 1Wire"));
  Serial.print(F("Les sondes 1Wire doivent être branchées sur le bus Digital "));
  Serial.println(F("2"));

  // Start up the library
  //Après cela, nous appellerons la méthode begin sur notre objet
DallasTemperature. Cette méthode sera responsable de l'initialisation du bus
OneWire.
  Temp_sensors.begin();



//////////////////////RESOLUTION EN BITS/////////////////////////////////////
// set the resolution to 9 bit (Each Dallas/Maxim device is capable of several
different resolutions)
```

```
  Temp_sensors.setResolution(12);

//Ses principales caractéristiques sont de changer ses numéros de bits en
fonction du changement de température.
//Par exemple, il change un bit dans 9. 10, 11, et 12 bits comme la température
change par pas de 0.5°C, 0.25°C, 0.125°C et 0.0625°C respectivement.
//Sa valeur de bits par défaut est 12 mais il change les valeurs selon le
changement de température.
//Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V, for long
distance, power is necessary.
//Converts Temperature to 12-Bit Digital Word in 750ms (Max)

////////////////////////////////////////////////////////////////////////////////




//Pour compter le nombre de capteurs découverts :
  Serial.print(F("Nombre de capteurs 1Wire détectés : "));
  Serial.println(Temp_sensors.getDeviceCount(), DEC);
  //Insertion dans la variable nbre_sensors
  nbre_Sensors = Temp_sensors.getDeviceCount();

  Serial.print(F("Nombre de capteurs DS18xx détectés : "));
  Serial.println(Temp_sensors.getDS18Count(), DEC);


//////////////////MODE ALIMENTATION PARASITE OU NORMAL
////////////////////////////
  // capteur en mode "parasite power" = la patte VCC du capteur alimentée avec le
signal
  Serial.print(F("Mode 'parasite power' est : "));
  if (Temp_sensors.isParasitePowerMode()) Serial.println(F("ON - (patte VDD du
capteur 1-Wire alimentée par le câble de signal DQ)"));
  else Serial.println(F("OFF - Ok branchement normal (5V sur patte VDD, et Signal
sur patte DQ du 1Wire"));

//Alimentation externe de la DS18B20
//
//Dans cette méthode, nous alimentons le DS18B20 par une méthode conventionnelle,
c'est-à-dire que la pate VDD est alimentée séparaément.
//Cette méthode est applicable pour des températures inférieures à +100 degrés
Celsius.
//Le principal avantage de cette méthode est qu'il n'y a pas de charge
supplémentaire sur la résistance utilisée dans cette méthode et qu'elle
fonctionne correctement.
//
//Mode d'alimentation parasite de la DS18B20
//
//Dans cette méthode, nous n'avons pas besoin d'une alimentation spéciale.
//Cette méthode est utilisée pour des températures supérieures à +100 Celsius.
//Dans une situation normale, cette méthode fournit un courant et une tension
efficaces à la DS18B20.
//Mais, lors d'un travail spécial, lorsque la DS18B20 convertit la valeur de la
température en numérique, la valeur du courant augmente jusqu'à une telle valeur
qui peut endommager la résistance.
//Pour limiter le courant et sauver la valeur et le bon fonctionnement du
DS18B20, il est nécessaire d'utiliser un mosfet pull up.
//Comme il est utilisé seulement pour une valeur de température spécifique, nous
utilisons une alimentation externe.

////////////////////////////////////////////////////////////////////////////////
/
```

```
////////////CALCULE DE LA TAILLE DU TABLEAU Sondes_1Wire Partie #2
///////////////
Serial.println();
Serial.print(F("Calcul du nombre de lignes du tableau Sondes_1Wire : "));
Serial.println(ROW_COUNT(Sondes_1Wire));

//////////////////////////////////////////////////////////////////////////////
///////



  //Ensuite, nous devons déclarer un tampon d'octets pour contenir l'ID du
capteur.
  //Puisque, comme nous l'avons déjà mentionné dans la section d'introduction,
les ID ont 64 bits, cela signifie que nous aurons besoin d'un tableau d'une
longueur de 8 octets.
  uint8_t address[8];

  int x = 0;


Serial.println(F("Initialisation du while Sondes_1wire[x]"));

  while (Temp_sensors.getAddress(address, x)) {
    Serial.print(F("Sonde N°"));
    Serial.print(x+1);
    Serial.print(F(" : "));
    Temp_sensors.getAddress(address, x);
    for (int i = 0; i < 8; i++) {
      Serial.print(address[i],HEX);
      Serial.print(F(" "));
    }
    Serial.println();

    x = x+1;
    // "x++" -> est identique à mon code, incrémente aussi la variable x (voir
dans la boucle "for (int i = 0...." )

  }




  Serial.println();

  Serial.println(F("-------Demande de température sur une sonde spécifique-------
"));




}



void reconnect() {
```

```
  // On boucle cette fonction jusqu'à la reconnexion au borker (voir si cela ne
pose pas d'autres soucis ?)
  while (!mqttClient.connected()) {
    // Tentative de reconnexion
    if (mqttClient.connect("Arduino","Arduino","Mot_de_passe")) {
      Serial.println();
      Serial.println("Reconnecté au broker !");
      // Once connected, publish an announcement...
      //mqttClient.publish("home-assistant/sensor01/brightness", "0");
      // ... and resubscribe
      mqttClient.subscribe("home-assistant/switch/Virtual_Switch_01/HA_Write");
    } else {
      Serial.print(F("Broker MQTT indisponible. Status : rc="));
      Serial.print(mqttClient.state());
      Serial.println(F(" , tentative de reconnexion dans 3 secondes..."));
      // Attente 3s pour rententer nouvelle connexion. attention aussi ici si on
a un watchdog qui tourne pour ne pas
      // foutre le souc (max 6s)
      delay(3000);
    }
  }
}









void loop() {

/*
 * Cette ligne de code va permettre de reconnecter l'arduino au broker en cas de
crash de celui-ci.
 * Typiquement lorsque le RPI redémarre ! (un redémarrage "Core" ou "Superviseur"
d'homeassistant n'est pas suffisant
 * pour déconnecter le broker (et c'est logique puisqu'il est installé sur le
Linux/RPI et pas comme surcouche
 * de Hoemassistant)
 */
 if (!mqttClient.connected()) {
    reconnect();
  }

  // This is needed at the top of the loop!
  //Apparemment cette commande est nécessaire au tout début du void loop ().
  mqttClient.loop();



  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  //Serial.print(F("Requesting temperatures..."));
  Temp_sensors.requestTemperatures(); // Send the command to get temperatures
  //Serial.println(F("DONE");
  // After we got the temperatures, we can print them here.


// Ensure that we are subscribed to the topic "MakerIOTopic"
/* On écoute le topic, via lequel Homeassistant va envoyer une commande ou une
info
 *  Donc ici, l'arduino, va se mettre à l'écoute du canal "home-
assistant/switch/Virtual_Switch_01/HA_Write"
 *  Ce canal est configuré dans le fichier confuiguration.yaml avec le reste de
la config du "capteur" à la ligne :
 *  "command_topic"
```

```
 *  C'est donc ce canal qui permettra à hoemassistant d'envoyer une valeur / une
commande à l'arduino
 *
 *  Du coup le cannal "HA_Read" permet à l'arduino d'envoyer un message / valeur
/ commande au broker MQTT et donc à Homeassistant
 *  et le canal "HA_Write" permet à Homeassistant/broker d'envoyer un message /
valeur / commande à l'arduino
 */
  mqttClient.subscribe("home-assistant/switch/Virtual_Switch_01/HA_Write");



  // Attempt to publish a value to the topic "MakerIOTopic"
/*Ici le nom du topic "home-assistant/david/humeur" est celui créé dans le
fichier de configuration "Sensors.yaml"
 * C'est donc le canal par lequel vons transiter les infos de ce capteur
("state_topic" dans le fichier de config de HA)
 *
 * Dans cet exemple, c'est un simple texte qui va venir sur lovelace afficher le
texte tout bêtement
 * Il n'y aura donc pas de modification du texte de ce message dans ce programme
!
 * Ici on envoie simplement le texte "Bonne !" vers HA.
 * Le "false" correspond à l'option "retained" du broker MQTT, cette option si
activée permet d'éviter les pertes
 * de paquet si j'ai bien compris.
 * Encore une fois pas sûr que cela soit bien utile en cas d'utilisation d'un
conexion réseau fiable (Ethernet ou fort
 * signal wifi)
 */
  if(mqttClient.publish("home-assistant/david/humeur", "Bonne !",false))
  {
    Serial.print(F("Publication message : '"));
    Serial.print(F("Test d'humeur'"));
    Serial.println(F(" avec success"));

  }
  else
  {
    Serial.println(F("Message non envoyé :("));
  }

/* Dans mon installation initiale, j'avais simplement connecté une
photorésistance au PIN A0 de l'arduino
 *  pour lire la valeur en mV de la luminosité
 *  Je mettais à jour cette valeur dans Homeassistant.
 */
  int sensorValue = analogRead(A0);
  Serial.print(F("Sensor value: "));
  Serial.print(sensorValue);
  Serial.println(F("mV"));
 /*
  * Ici le itoa convertit une valeur entière (int) en un texte.
  * Chercher "itoa" dans l'aide de C++
  *
  * Dans mon cas, ma valeur est un entier entre 0 et 1024, comme toutes les
valeurs analogiques de
  * l'arduino, donc pas de soucis, par contre on va voir pour les sondes de
température 1Wire qu'il faudra multiplier
  * par 100 pour avoir un entier et refaire une division sous HA pour afficher la
bonne valeur
  */
  mqttClient.publish("home-assistant/sensor01/brightness", itoa(sensorValue, buf,
10));
```

```
///// Nombre de lignes dans la tableau Sondes_11Wire:
int nbre_lignes_tableau_1Wire = ROW_COUNT(Sondes_1Wire);

float Tempe = 20.0;
Tempe = Temp_sensors.getTempC(Sondes_1Wire[2]);



/// Affichage des températures dans une boucle.
  //while (x < 5) {
    for (int x = 0; x <= nbre_lignes_tableau_1Wire ; x++) {
      if (Temp_sensors.isConnected(Sondes_1Wire[x])){
    Serial.print(F("Sonde N°"));
    Serial.print(x+1);
    Serial.print(F(" : "));
    Serial.print(Temp_sensors.getTempC(Sondes_1Wire[x]));
    Tempe = Temp_sensors.getTempC(Sondes_1Wire[x]);
    Serial.println(F("°C"));

  //Ici je multiplie la valeur de température par 100 pour m'éviter le tronquage
de mon float (garde juste la partie entière!)
  Tempe = Tempe * 100;

    //Ici le nom du topic "home-assistant/david/humeur" est celui créé dans le
fichier de configuration "Sensors.yaml"
     if(mqttClient.publish("home-assistant/sensor01/1Wire_03", itoa(Tempe, buf,
10)))
     //if(mqttClient.publish("home-assistant/sensor01/1Wire_03", "1856"))
      {
      Serial.println(F("Publication de la Température avec success"));

      }
    else
    {
      Serial.println(F("Could not send message :("));
    }

    }
  }

  Serial.println();



  // Dont overload the server!
 /*Si watchdog à 8s, le delay max doit être de 6 secondes ! Et attention, car
cela doit comprendre tous les delays parsemés
 * dans le tout le code ....
 *
 *  Ne pas rafraichir trop vite (le tauc de rafraichissement le plus rapide que
j'ai trouvé était toujours
 *  d'environ 1 seconde
 */
  delay(4000);
/**
 * C'est le watchdog, ici il est réglé à 8s, c'est à dire que si après 8s cette
commande n'est pas lancée et que
 * donc le wtachdog n'est pas resetté dans le cas le CPU redémarre !
 * Alors attention à ne pas monter trop haut les delay du programmes !
 */
   wdt_reset();

}
```

Désolé, j'ai pas eu la motivation pour me taper toute la colorisation syntaxique, j'ai juste coloré les commentaires en gris que vous voyez mieux ressortir le « code utile ».

Bon le code est assez commenté/documenté, cela devrait être suffisant, mais on va quand même voir rapidement les parties importantes.

```
////********** Librairie pour le wtachdog//////////
#include <avr/wdt.h> // Version utilisée : Pas de N° de version trouvée !!
//*******************************
```

Le watchdog permet à l'arduino de redémarrer de lui-même s'il y a un souci.
En gros, en cas de blocage et après X secondes (dans ce programme j'ai réglé au maximum c'est-à-dire 8 secondes) l'arduino va redémarrer de lui-même. C'est l'équivalent d'un appui sur le bouton « Reset » de la carte.

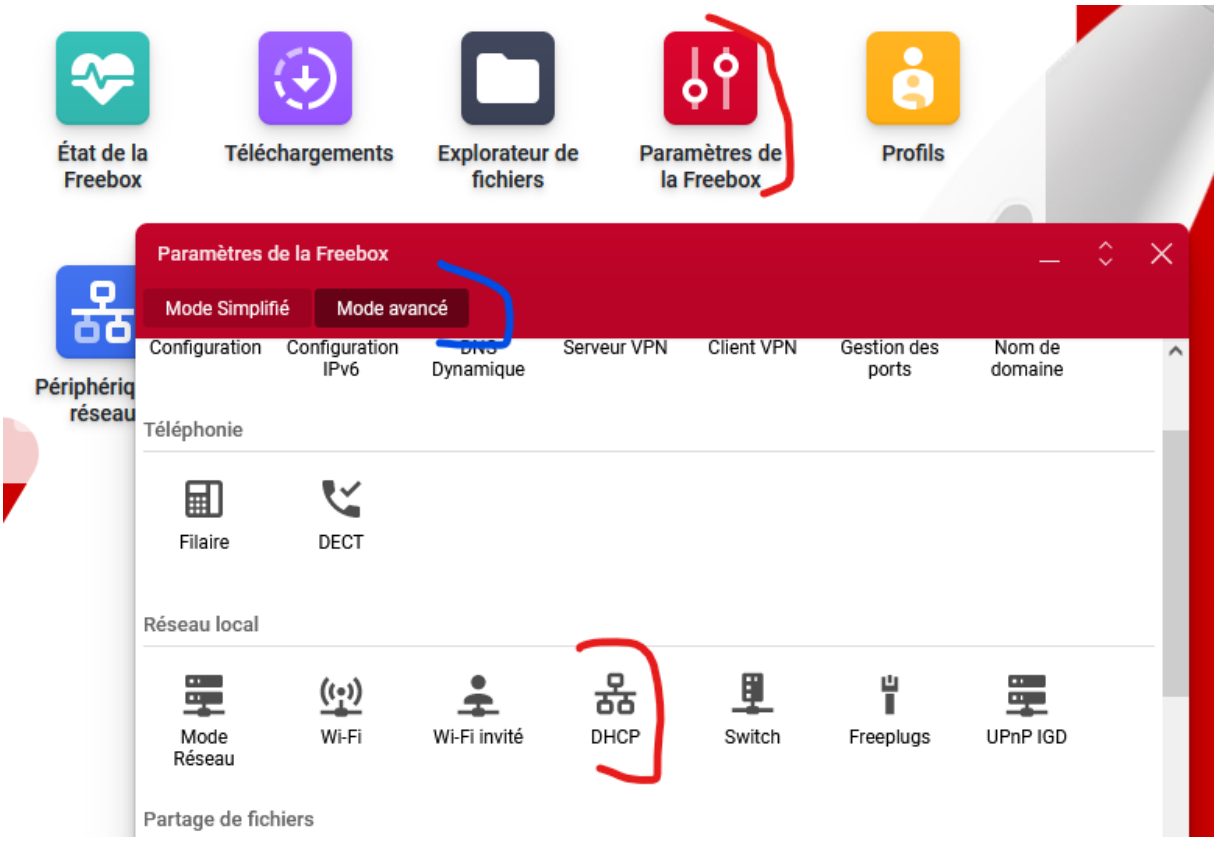
```
////////////CALCUL DE LA TAILLE DU TABLEAU Sondes_1Wire Partie #1 ///////////////
#define ROW_COUNT(array)    (sizeof(array) / sizeof(*array))
#define COLUMN_COUNT(array) (sizeof(array) / (sizeof(**array) * ROW_COUNT(array)))
////////////////////////////////////////////////////////////////////////////////
```

Ce code me permet de calculer la taille du tableau dans lequel je vais stocker mes numéros de sondes 1Wire. En effet, chaque sonde a un numéro unique pour s'assurer que lors de la communication de plusieurs sondes sur le même réseau, il n'y a pas de risque « de mélange de données ».

```
// Définition de l'adresse du broket MQTT (configuration d'un "bail statique" sur la box !) : Freebox -> Paramètres de la freebox -> DHCP
const char* MQTTserver = "192.168.1.50";
```

Ici, c'est l'adresse IP du serveur MQTT (donc dans mon exemple, l'adresse IP du Raspeberry).
Pour éviter un bordel à chaque redémarrage, j'ai défini une IP fixe pour mon raspberry (pour mon arduino aussi d'ailleurs !) directement dans ma freebox.

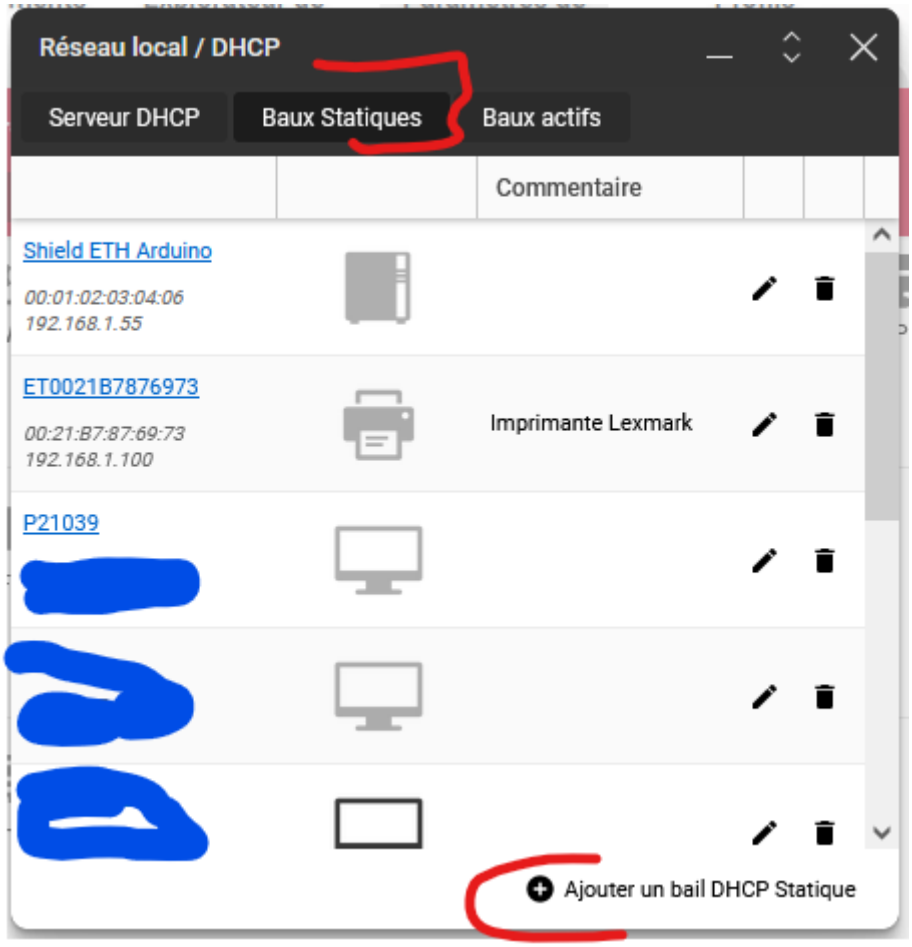
Sur freebox, taper dans le navigateur http://mafreebox.freebox.fr/, puis ouvrir « paramètres de la freebox ». Choisir le « mode avancé » (en haut de la fenêtre) et finalement double cliquer sur « DHCP »

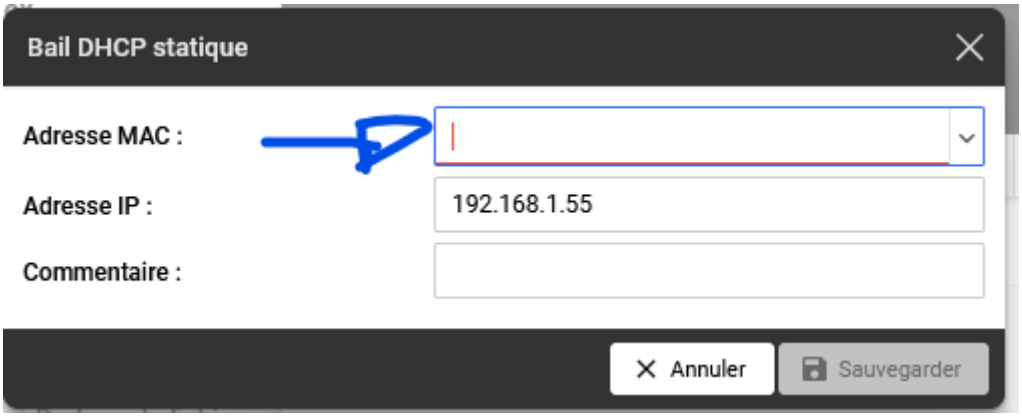


Dans la fenêtre qui vient de s'ouvrir, cliquer sur l'onglet « Baux statiques », on voit une liste de périphériques chez moi, dont mon Arduino (tout en haut de la liste).

⚠ Attention, de base ma carte shield ehternet n'avait pas d'adresse MAC de définie !
Pour info : L'adresse Mac c'est le même principe que le numéro de série des sondes 1Wire, c'est un numéro unique qui permet d'éviter de confondre 2 périphériques réseaux sur un réseau !
Si vous voulez changer les paramètres sur votre box, je vous conseille, de démarrer la première fois votre carte arduino avec le code fourni (ouverture du fichier .ino puis cliquer sur « Téléverser ») et seulement ensuite de faire les manip décrites ici.

Puis cliquer sur le bouton du bas « Ajouter un bail DHCP statique »





Pour le raspeberry : Taper l'adresse IP de votre choix (dans notre exemple 192.168.1.50).

Puis ouvrir la liste déroulante « Adresse MAC », vous devriez trouver votre Arduino avec son nom (Raspberyy / Raspeberry fondation). Le sélectionner puis cliquer sur sauvegarder.

Pour l'arduino : Taper l'adresse IP de votre choix (dans notre exemple 192.168.1.55).

Puis ouvrir la liste déroulante « Adresse MAC », vous devriez trouver votre Arduino avec l'adresse MAC définie dans le programme (00 :01 :02 :03 :04 :06). Sélectionner puis sauvegarder.

Au prochain reset (redémarrage) de l'arduino il aura l'adresse IP définie ici !

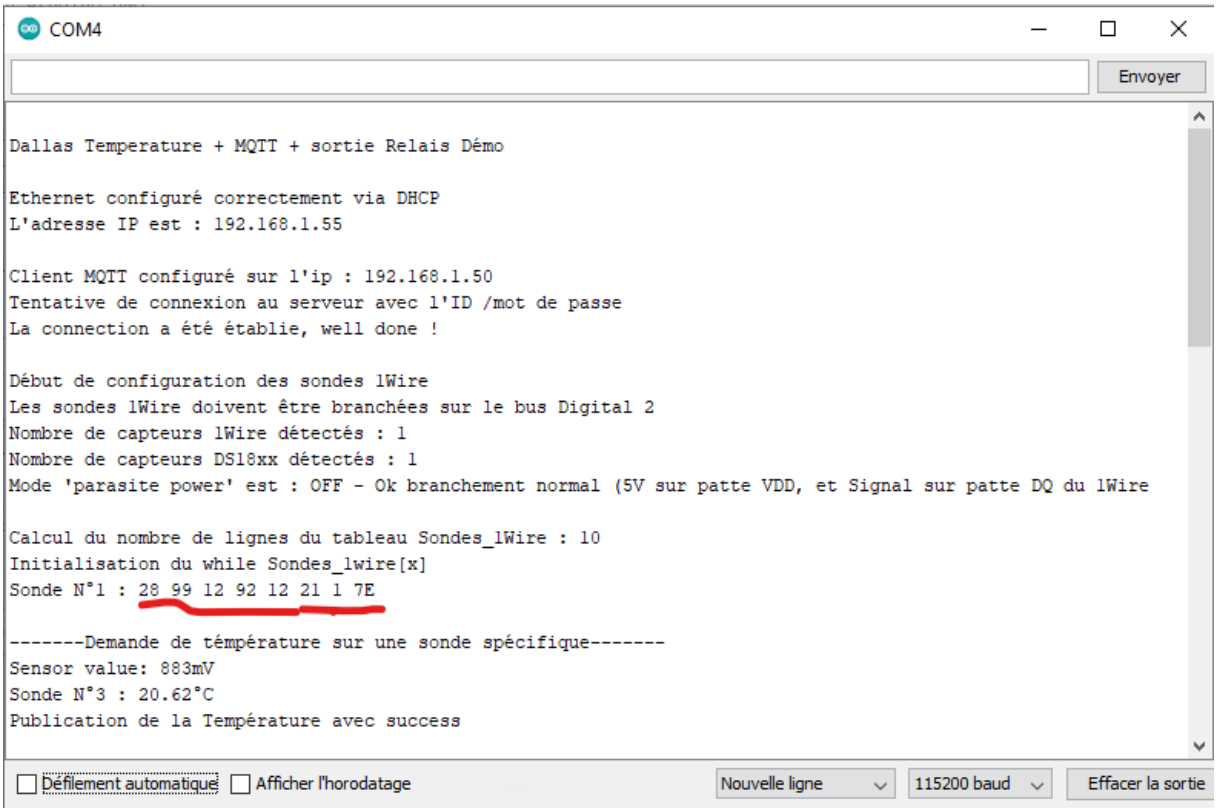
```
DeviceAddress Sondes_1Wire[] = {
  { 0x28, 0x20, 0xEA, 0xEA, 0x17, 0x20, 0x6 , 0xB1 },   //Sonde 01
  { 0x28, 0xE1, 0xB6, 0xF1, 0x17, 0x20, 0x6 , 0x1 },    //Sonde 02
  { 0x28, 0x99, 0x12, 0x92, 0x12, 0x21, 0x1 , 0x7E },    //Sonde 03
  { 0x28, 0x24, 0x22, 0x1F, 0x18, 0x20, 0x6 , 0xDE },   //Sonde 04
  { 0x28, 0x19, 0xDA, 0x39, 0x12, 0x21, 0x1 , 0xB5 },   //Sonde 05
  { 0x28, 0x5A, 0x6D, 0x1C, 0x18, 0x20, 0x6 , 0x5B },   //Sonde 06
  { 0x28, 0x69, 0x54, 0xF, 0x18, 0x20, 0x6 , 0x28 },    //Sonde 07
  { 0x28, 0x1B, 0x73, 0xDF, 0x17, 0x20, 0x6 , 0x29 },   //Sonde 08
  { 0x28, 0x36, 0x53, 0xE6, 0x17, 0x20, 0x6 , 0xF6 },   //Sonde 09
  { 0x28, 0x63, 0xBA, 0x8, 0x18, 0x20, 0x6 , 0x19 }     //Sonde 10
};
```

Dans ce tableau j'ai entré manuellement les numéros (hexadécimaux) de série de mes sondes.
Vous verrez, dans la suite du programme, j'écris dans la console série les numéros des sondes qui sont connectées à l'arduino.
Ce que j'ai donc fait, j'ai scotché sur chaque sonde un papier avec 01, 02, 03, etc…
Et ensuite j'ai connecté une sonde après l'autre, et j'ai noté la relation entre mon numéro scotché sur chaque sonde (01, 02 etc…) avec son numéro de série interne.

Je sais donc que la sonde 03 est connectée au moment de faire les tests.

Dans le tableau ci-dessus la sonde 03 porte le numéro « 0x28, 0x99, 0x12, 0x92, 0x12, 0x21, 0x1, 0x7E ». Dans la console les *0x* n'apparaitront pas, on retrouvera donc pour cette même sonde cette série de numéro hexadécimaux : « 28 99 12 92 12 21 1 7E », comme sur la copie d'écran ci-dessous.

Donc n'oubliez pas de rajouter les *0x* devant chaque numéro hexadécimal quand vous remplirez le tableau avec les numéros de vos propres sondes.



```
void callback(char* topic, byte* payload, unsigned int length)

String Str;

  Serial.print(F("**** Message arrivé depuis MQTT broker **** -> [ "));
  //Imprime le chemin sur le broker
  Serial.print(topic);
  Serial.print(F(" ] -> ' "));
  for (int i=0;i<length;i++) {//ici le lenght c'est la variable de la fonction callback
```

C'est la fonction (void callback) la plus importante si vous voulez réceptionner des messages MQTT !

En fait l'arduino écoutera tous les messages qui viennent d'homeassistant (mais aussi des autres clients MQTT que vous pourriez installer plus tard) grâce à cette fonction. Il faut donc s'assurer que nous ne traitons que les messages qui sont destinés à l'Arduino (on va voir ça juste en dessous).

A partir de la deuxième ligne, on copie caractère par caractère le contenu du message/canal dans la variable de type String (= Texte) nommée « Str ».

```
char* Canal_Switch_Virtuel = "home-assistant/switch/Virtual_Switch_01/HA_Write";

int valeur_Canal = strcmp(topic, Canal_Switch_Virtuel);
```

La première ligne va définir quel canal (ou quel nom de message) nous intéresse.
Dans mon programme, c'est le message/canal « home-assistant/switch/Virtual_Switch_01/HA_Write » qui m'intéresse et pas un autre, puisque c'est celui-là que j'ai défini pour donner l'ordre d'activer le relai connecté à la carte Arduino.

Évidemment, lorsque vous aurez d'autres relais à activer, vous aurez un autre canal à surveiller (donc création de la nouvelle entité sous Homeassistant et copier/coller de ces lignes + adaptation pour l'ajouter).

La deuxième ligne va comparer caractère par caractères le nom du message arrivé, si les 2 valeurs sont strictement identiques, le résultat de la fonction « strcmp » est un 0.

```
  if(valeur_Canal == 0 && Str == "1"){
    Serial.println(F("Bouton appuyé, envoie d'un 1 -> Bouton INACTIF sous HomeAssistant"));
    mqttClient.publish("home-assistant/switch/Virtual_Switch_01/HA_Read", "1");
    digitalWrite(8,HIGH);
    Serial.println(F("Relais ouvert"));
```

Dans la suite de la fonction callback, on va réaliser les actions en fonction des messages que l'on va recevoir.

Ici ma condition « If » vérifie que j'ai bien une valeur de 0 pour ma variable « valeur_canal » et que j'ai le message « 1 » dans ma variable « Str » (et qui vient de « Payload » -> une variable de la

fonction callback).
Dans ce cas j'effectue 2 actions :

1. J'envoie un retour à homeassistant pour allumer le bouton sur l'interface grâce à : mqttClient.publish(….
2. J'active le relais branché sur le pin 8 de l'arduino.

J'ai un code identique que j'utilise quand je trouve une valeur « 0 » dans « Str » (et dans ce cas je désactive le relais et envoie le message à homeassitant.

```
 pinMode(8,OUTPUT);
 //Désactivation du relais (chez moi le relais de la carte (Elegoo - 8 channel 5V Relay module) est inversé
 digitalWrite(8,HIGH);
```

Ici, je défini le numéro de Pin vers lequel va pointer la commande du relais (Signal -> Câble gris voir page 3).

Puis j'active (initialise) la sortie qui correspond à la position inactive du relais (= LED sur le relais éteinte).

```
  wdt_enable(WDTO_8S);
```

C'est le réglage du watchdog à 8 secondes (le maximum possible). Après 8 sec si le signal n'est pas reseté l'ardunio redémarre de lui-même.

```
 if (Ethernet.begin(mac) == 0) {
  Serial.println(F("Impossible de configurer Ethernet via serveur DHCP !"));
  for (;;);
 }
```

Dans beaucoup d'exemples que j'ai trouvé sur internet, l'adresse IP est entrée en dur dans le code de programmation, mais rien n'empêche le serveur DHCP d'avoir déjà attribué l'adresse IP à un autre périphérique.

Donc pour pouvoir utiliser le bail statique que l'on a défini dans la freebox juste avant, il faut utiliser l'attribution de l'adresse IP par le serveur DHCP. C'est tout simplement ce que cette portion du code fait.

```
mqttClient.setServer(MQTTserver, 1883);
```

Ici on va définir l'adresse du serveur MQTT et son port.
L'adresse est défini par la variable « MQTTserver », et dont la valeur est dans notre tuto de « 192.168.1.50 » et qui correspond à l'adresse IP du Raspberry sur lequel le broker MQTT est installé.

```
if (mqttClient.connect("Arduino","Arduino","Mot_de_passe"))
 {
  Serial.println(F("La connection a été établie, well done !"));
 }
 else
 {
  Serial.println(F("Il semblerait que le serveur ne réponde pas..."));
 }
```

Ici on entre les paramètres identifiants et mot de passe à la fonction mqttClient.connect.

1er argument = ID (je ne sais pas ce que cela fait, je le laisse identique à l'identifiant)

2ème argument = user - C'est le nom d'utilisateur et donc celui que nous avons défini sous homeassistant dans la configuration de Mosquitto broker (voir page 6 pour rappel) et qui est « Arduino » dans cet exemple.

3ème argument = pass – C'est le mot de passe (idem voir page 6), dans notre exemple est « Mot_de_passe »

```
mqttClient.setCallback(callback);
```

On défini que la fonction callback du client MQTT est celle que nous avons vu avant (void callback).

```
 Temp_sensors.begin();
  Temp_sensors.setResolution(12);
```

Ici on initialise les sondes de température 1Wire, puis on les règle sur leur sensibilité la plus élevée (1 incrément vaut en mode 12 bits 0.0625°C).
Dans cette résolution le temps de mesure + conversion est plus long (environ 750ms) attention donc à ne pas appeler trop fréquemment les valeurs de la sonde, sous peine d'avoir des valeurs « sympathiques » 😉.

```
 while (Temp_sensors.getAddress(address, x)) {
  Serial.print(F("Sonde N°"));
  Serial.print(x+1);
  Serial.print(F(" : "));
  Temp_sensors.getAddress(address, x);
  for (int i = 0; i < 8; i++) {
   Serial.print(address[i],HEX);
   Serial.print(F(" "));
  }
  Serial.println();

  x = x+1;
 }
```

Cette fonction, me permet de compter le nombre de sondes réellement connectées, et me les affiche dans la console série (utile en phase de test, moins en production).

```
void reconnect()
  while (!mqttClient.connected()) {
    // Tentative de reconnexion
    if (mqttClient.connect("Arduino","Arduino","Mot_de_passe")) {
      Serial.println();
      Serial.println("Reconnecté au broker !");
```

Cette fonction, nous permet de nous reconnecter automatiquement en cas de coupure.
Elle est utile lorsque le raspberry redémarre, ou lors d'une perte de réseau internet.

Notez que vous avez à nouveau l'identifiant et mot de passe dans cette section, à modifier donc, si vous n'utilisez pas les mêmes paramètres dans vos réglages de mosquitto broker sous homeassistant.

```
void loop() {

if (!mqttClient.connected()) {
    reconnect();
  }

mqttClient.loop();
```

Ici on arrive dans la partie du code principale.
Les 2 premières lignes sont essentielles, la première qui vérifie à chaque itération de la fonction « loop » que l'arduino est connecté au broker. Si ce n'est pas le cas on appelle la fonction reconnect qu'on a vu juste avant.
La seconde je ne sais pas à quoi elle sert, mais sans elle cela ne fonctionne apparemment pas !

```
if(mqttClient.publish("home-assistant/david/humeur", "Bonne !",false))
```

C'était mon premier test pour envoyer un message de l'arduino vers le broker (et donc vers hoemassistant). On voit ici que « home-assistant/david/humeur » correspond à mon « state_topic » sous homeassistant.

```
for (int x = 0; x <= nbre_lignes_tableau_1Wire ; x++) {

if (Temp_sensors.isConnected(Sondes_1Wire[x])){
Tempe = Temp_sensors.getTempC(Sondes_1Wire[x]);

Tempe = Tempe * 100;
```

```
if(mqttClient.publish("home-assistant/sensor01/1Wire_03", itoa(Tempe, buf, 10)))
```

Ici je teste toutes les sondes de mon tableau.
Chaque sonde qui est connectée (= disponible), on vient enregistrer sa valeur dans la variable « Tempe » via l'appel « Temp_sensors.getTempC(… »

Puis on va multiplier la valeur de température par 100. En effet je ne suis pas arrivé à transmettre une valeur décimale via MQTT à homeassistant (sinon il ne prend que la valeur entière, c'est un peu dommage pour une sonde qui mesure aussi finement que 0.0625° !).
Du coup pour éviter ce souci, je multiplie par 100 côté arduino, puis je divise la valeur transférée par 100 du côté de Homeassistant.

« Itoa » est une routine de conversion d'un chiffre entier vers une chaine de caractère. Ceci est nécessaire pour faire transiter les infos vers le broker/homeassistant.

```
  wdt_reset();
```

C'est mon signal de réinitialisation du Watchdog (chien de garde en français, mais perso je préfère la version anglaise ! 😉).
En fait si vous vous souvenez, on a configuré notre watchdog à 8 secondes.
Ce qui fait que la commande « wdt_reset » doit être activée avant la fin de ces 8 secondes.
Donc attention, aux « delay » éparpillés dans le code, s'ils comptabilisent plus de 8 sec vous allez redémarrer votre arduino sans fin !

Une amélioration du code, pourrait être ne pas utiliser la commande delay (qui fige complètement l'arduino), mais plutôt une routine qui teste le nombre de millisecondes entre 2 appels.

OUF C'EST TERMINÉ !

EN ESPÉRANT QU'IL AIT SERVI AIT QUELQUE CHOSE :-) !