Open Thermostat

Travail sur le thème de l'I.O.T, objet connecté gérant des pompes à chaleur

Étude des classes cpp

Projet Tuteuré

Analyse et modélisation du code existant

Lenoir Adrien - Louveau Benoît - Mahiou Yannis

Raffray Elouan - Salabert Gabin

Décembre 2017

1 Application.cpp

1.1 Documentation

La classe Application fait référence à celle du même nom en Androïd. Elle représente le contexte de l'application, et permet de maintenir son état global. C'est dans cette classe qu'est répertoriée la configuration de l'application, c'est à dire les logs et toutes les variables necessaires (celles des capteurs, de la wifi, etc..) qui doivent être initialisées et sauvegardées.

1.2 Application::Application() et Application::Application(String s)

2 constructeurs différents (avec ou sans fichier).

1.3 void Application::init()

Initialisation des variables des variables à partir d'un fichier (ou non). Spiffs est un système de fichier pratique étant données les contraintes de la carte telles que sa RAM limitée par exemple. Il ne prend pas en charge les répertoire, et stocke juste une liste plate de fichiers mais accepte les "/"dans les noms de fichiers.

1.4 void Application::load()

Charge le fichier en mémoire.

1.5 void Application::save()

Sauvegarde les données actuelles dans un nouveau fichier qui remplace le précédent.

1.6 void Application::AddLog(String s)

uint8_t = unsigned$_c hardevaleurmaximale 255. Ajouteunlogavecl'heureactueldanslecycle. Cyclegrparuneboucleetunmodulopourne$

2 SensorManager.cpp

2.1 Documentation

Cette classe permet d'initialiser les capteurs DHT et Présence tout en controlant en permanence les données récupérées par la lecture. Elle permet aussi d'avoir accès directement aux valeurs humidité et température. Elle ajoute aussi le nombre de présence détectées aux logs et patiente entre chaque lecture.

2.2 SensorManager::SensorManager() et SensorManager::SensorManager(Application *c, DomoticzBroadcaster * domo)

Constructeur par défaut et constructeur à partir de configurations existantes.

2.3 uint8_t SensorManager::getStatePIR()

Équivalent d'un getter, retourne l'état de l'IR.

2.4 void SensorManager::init()

Initialisation de la température et de l'humidité par défaut aini que de leur capteur.

2.5 void SensorManager::ReadTempHum()

Lecture des valeurs humidité et température. Retourne la chaine formatée des valeurs sondées. Si le délai entre deux lectures a été respecté alors on lis. Vérification de la lecture : si les valeurs ne sont pas des nombres alors on retourne une erreur.

2.6 float SensorManager::getDirectTemp()

Getter qui retourne la chaine formatée de la température.

2.7 float SensorManager::getDirectHum()

Getter qui retourne la chaine formatée de l'humidité.

2.8 void SensorManager::readPir(uint8_t prestime)

Lecture du capteur de présence. Si le délai entre la dernière et la nouvelle lecture a été respecté ou que le capteur est à zero. Alors on lit le capteur. Le delai permet de laisser le temps de lire qu'un mouvement est détecté par le capteur.

3 WifiManager.cpp

3.1 Documentation

Cette classe se concentre sur l'initialisation de la classe WIFI Arduino.

3.2 void initAP()

Déclare l'ip, la gateway, le masque de sous-réseau, l'adresse MAC et configure le point d'accès Wifi en softAP (= routeur). Nomme le point d'accès ESP8266 + adresse MAC avec le mot de passe "password".

3.3 bool connexion(char *ssid, char *pwd, int expiration)

Parse les IP, Gateway et DNS de l'Application, configure le wifi avec ces infos et tente la connexion. Tant que la connexion n'a pas réussi ou que le timeout n'est pas atteinte, on fait clignoter la LED pour signifier à l'utilisateur que la connexion est en cours. Si la connexion réussit, on allume définitivement la LED sinon on l'éteint. Enfin on renvoie si oui ou non la connexion a réussi.

3.4 String ipToString(IPAddress ip)

Mets en forme texte l'objet IPAddress. Fait office de toString().

3.5 void parseBytes(const char* str, char sep, byte* bytes, int maxBytes, int base)

Fait office de parseur d'adresse IP, Gateway, DNS vers un tableau de byte. On passe l'adresse à convertir, le séparateur '.', le tableau de byte qu'on souhaite récupérer par adresse, le nombre max de byte du tableau et enfin la base vers laquelle on souhaite convertir. Pour i allant de 0 à maxBytes, on convertit dans une certaine base (10 souvent), on relit la chaîne jusqu'au séparateur, si on est à NULL ou '' on stop ici sinon on passe au premier caractère après le séparateur.

3.6 WifiManager(Application *c)

Constructeur de WifiManager, initialise lconfig aux données de l'Application passée en paramètre.

3.7 void initWifi()

Vérifie la présence d'un fichier de sauvegarde qui certifie que le WIFI a été initialisé. Si non, il appelle InitAP(), puis il convertit en char* toutes les Strings pour permettre la lecture caractère par caractère. Il teste si connexion(ssid, pwd, 20) renvoie true, si oui il ajoute la sauvegarde. Sinon il demande la reconfiguration du WIFI par l'utilisateur.

4 IRManager.cpp

4.1 IRManager()

Constructeur qui initialise l'attribut irSender avec l'aide de la classe IRSenderBigBang, qui reçoit l'attribut Application *lconfig->irPin.

4.2 IRManager(Application *c)

Initialise la configuration de IRManager par l'application.

4.3 void init(String server, int zone)

La méthode init initialise le IRSender. Ce constructeur est utilisé pour l'IDE Arduino.

4.4 uint8_t getHPState()

Les getter retournent l'état de la pompe (0 éteinte ou 1) ou la température de la pompe.
Retourne result, qui vaut 1 si l'envoi d'IR marche, 0 sinon.

4.5 uint8_t getHPTemp()

Si l'utilisateur envoie des valeurs valides, cette méthode initialise un tableau de modèles de pompes à chaleur.

4.6 uint8_t IRManager::sendIR(String hpmodel, String power, String fmode, String fan, String temp, String vair, String hair)

Ce tableau est ensuite parcouru: on cherche le modèle de pompe à chaleur entré par l'utilisateur. On fait ensuite appel à la méthode send() qui envoie un signal IR avec les paramètres spécifiés par l'utilisateur.

5 DomoticZ.cpp

Envoie l'état (1 ou 0) du capteur de présence au serveur Domoticz. Envoie la température (passée en paramètre) au serveur Domoticz. Change le mode de fonctionnement (passée en paramètre) de la pompe selon l'argument passé en paramètre. Retourne la température en parsant le résultat obtenu par le serveur Domoticz.

6 TimeNTP.cpp

6.1 TimeNtp()

Constructeur vide.

6.2 TimeNtp(String server, int zone)

Constructeur initialisant les attributs NtpServerName et timeZone avec ses paramètres.

6.3 void init(String server, int zone)

Initialise l'objet pour le rendre fonctionnel. Initialisation du serveur, de la timeZone et du tempo auquel récupérer l'heure).

6.4 time_t getTime(void)

Retourne le temps en secondes depuis le 1er Janvier 1970.

6.5 time_t globalGetNTPTime()

Retourne le résultat de getNtpTime().

6.6 time_t globalGetNTPTime();

Récupère l'heure sous forme d'un paquet depuis un serveur ntp (avec un timeout de 7sec), puis l'ajuste avant de la retourner. L'ajustement est fait en plusieurs étapes. Le temps NTP est le nombre de secondes écoulée depuis 1900. Or le temps UNIX commence le 1er Janvier 1970, on enlève donc 70ans en seconde a la date NTP (2208988800 secondes). La timeZone est ensuite appliquée, suivie de l'heure d'été si présente. Retourne 0 si echec.

6.7 void sendNTPpacket(IPAddress address)

Efface le buffer du paquet, puis le rempli selon le contenu d'un paquet NTP. Ce paquet est ensuite envoyé à l'adresse précisée en paramètre.

6.8 bool summertime(int year, byte month, byte day, byte hour, byte tzHours)

Retourne true si l'heure actuelle est en heure d'été et false sinon.

7 Web.cpp

7.1 Web(Application *c,SensorManager *s, Thermostat * t,WifiManager * lw, IRManager *ir, TimeNtp * ti, DomoticzBroadcaster *dom) : server(80)

Initialisation des paramètres lconfig, lsensor, lthermos, lwifi, lIR, ltimentp et ldomo avec les attributs passés en paramètre.

7.2 void sendNTPpacket(IPAddress address)

Démarrage du serveur web. Initialisation de la page de lecture de la température et de l'humidité. Reboot. Initialisation de la page de saisie et lecture de la valeur de l'étalonnage. Requête du statut du capteur de présence. Envoi d'une trame Infrarouge. On en mode Eco. Off. Initialisation de la page racine du service Web. Initialisation de la page de lecture et de modification des valeurs de connexion Wifi. Log page. config page. sensor page. formulaire config. formulaire mode racine. Initialisation de la page de config planning.
Formulaire Domoticz: -Dombroadcaster -Domoticz save -Check password form -Formulaire addplanning -Delete planning

7.3 void handle_root()

Page racine. Gestion de la connexion.

7.4 void sensor_print()

Page Capteur.

7.5 void sendir()

Récupération des valeurs saisies pour l'IR et envoie lorsque l'utilisateur fait appel au service sendir.

7.6 void modee()

Association des valeurs et des variables pour le mode de fonctionnement en fonction de ce que l'utilisateur a entré sur la page web.

7.7 void wifiCredentials()

Quand l'utilisateur fait appel au service /wificredentials.

7.8 void calibrateT()

Quand l'utilisateur fait appel au service /calibrateTemp.

7.9 void calibrateH()

Quand l'utilisateur fait appel au service /calibrateHum.

7.10 void saveconfig()

Mise en mémoire de tous les champs saisis pour la configuration.

7.11 void delplanning()

Modification/Suppression planning.

7.12 void saveplanning()

Mise en mémoire des champs saisies.

7.13 void savedomo()

Mise en mémoire des champs saisies.

7.14 void logging()

Gestion de la connexion.

8 Thermostat.cpp

8.1 Thermostat(String cal, String s, Application *c, SensorManager *se, IRManager *ir,Domoticz *domo)

Initialisation des paramètres lconfig, lsensor, lIR, file, fileCal, ldomo avec les attributs passés en paramètre.

8.2 uint8_t IsStarted(void)

Retourne la variable checkConfort.

8.3 void init(void)

Si tableau Cal est vide désactivation, libération mémoire et suppression de AlarmTab Lecture du fichier file si échec création de file avec des valeurs par défaut sinon assotions des valeurs aux variables. Appel de load_Cal(). Création ou mise à NULL du tableau AlarmTab Appel de setAlarm en fonction des valeurs du tableau Cal. Fait une action en fonction du mode de fonctionnement.

8.4 void setAlarm(uint8_t i,char mode, uint8_t nbday,uint8_t h,uint8_t s)

Réglage de l'alarme : Celons les différent mode Off = O, Eco = E, Comfort = C, Frostfree = F dans un switch Pour toute la semaine si nbday = 0 Sinon pour le jour indiquer. Default erreur alarme

8.5 void load_Cal()

Chargement du fichier fileCal.

8.6 void save_Cal(int l)

Sauvegarde le tableau Cal dans le fichier fileCal sans la ligne l correspondant au tableau Calidx. Si le tableau Cal n'est pas vide on le supprime puis on appelle $\text{load}_c aletenfinonappelinit()$.

8.7 void save_Cal(String n)

Création d'une chaine de string formaté des valeurs Eco, Frostfree, Comfort, hysteresis, triggerHeat, prestime, presentbool, presmin, presmax. Suppression de l'ancien fichier file. Sauvegarde de la chine de caractères dans le fichier file.

8.8 void globalEco()

Enveloppe le texte « Eco » dans la variable modeCal pour setSyncProvider. Puis si ldmo n'est pas NULL appel la fonction SendMode(de DomoticzBrodcaster.cpp) avec modeCal pour paramètre.

8.9 void globalOff()

Enveloppe le texte « Off » dans la variable modeCal pour setSyncProvider. Puis si ldmo n'est pas NULL appel la fonction SendMode(de DomoticzBrodcaster.cpp) avec modeCal pour paramètre.

8.10 void globalComfort()

Enveloppe le texte « Comfort » dans la variable modeCal pour setSyncProvider. Puis si ldmo n'est pas NULL appel la fonction SendMode(de DomoticzBrodcaster.cpp) avec modeCal pour paramètre.

8.11 void globalFrostfree()

Enveloppe le texte « Frostfree » dans la variable modeCal pour setSyncProvider. Puis si ldmo n'est pas NULL appel la fonction SendMode(de DomoticzBrodcaster.cpp) avec modeCal pour paramètre.

8.12 void run() //main method

Récupération de la température (via capteur ou Domoticz, l'état et la température de la pompe. Récupération du mode. Récupération de la température attendue celons le mode.

Si la température réelle est < = 0: on return.
Si une présence est détectée et que toutes les conditions sont réunies passage de la température attendue en Comfort.
Si mode OFF et pompe éteinte et (température actuelle <= température attendu – hystérésis): pompe ON.
Si le mode est Off que l'état de la pompe est 1: on envoie la commande IR pour éteindre la pompe.
Si la température réelle >= température attendue + trigerHeat: on envoie la commande IR pour éteindre la pompe.
Si l'état de la pompe est 1 et (tempPompe > tempAttendue+hysterésis): on réduit la température de la pompe.
Si l'état de la pompe est 1 et (tempPompe < tempAttendue-hysterésis): on augmente la température de la pompe.

8.13 void off()

Eteint la pompe et envoie le mode à Domoticz.

8.14 void on()

Envoie le mode à Domoticz.

8.15 void Autochecking(void)

Si toutes les conditions sont réunies: Récupération des valeurs du tableau puis définition du mode en fonction des valeurs.