```
/*
*************************************************************************************************************************
**********************
Projet Centrale domotique DomoMaison 2017 pour carte Arduino Mega 2560 et carte Ethernet (W5100)
Version 1.70 du 06/09/2017
Copyright 2017 PRIN Jean-Luc
Ce programme fonctionne avec le protocole Websockets (HTML5).
*/

// -------------------------------------------------------------------------------------------------------------------
----------------------
// ********************************************** ENTETE DECLARATRICE
**************************************************************************
// -------------------------------------------------------------------------------------------------------------------
----------------------
// Dans cette section, sont déclarées les librairies, les constantes et les variables
// ------------------------------------------- Section librairie et Définitions ----------------------------------------
----------------------
// --- Inclusion des librairies  de base nécessaire au système ---
//#include <Arduino.h>
#include <Wire.h>                   // librairie I2C
#include <DS1307RTC.h>              // librairie pour RTC DS1307
#include <LiquidCrystal.h>          // librairie pour afficheur LCD
#include <SPI.h>                    // librairie SPI - obligatoire avec librairie Ethernet
#include <Ethernet.h>               // librairie Ethernet
#include <EthernetUdp.h>
#include <Time.h>                   // librairie pour la gestion du temps
#include <SD.h>                     // librairie pour gestion de la carte SD
#include <WebSocket.h>              // librairie nécessaire pour le protocole WebSocket
#include <limits.h>                    // Informations sur les constantes de compilation
#include <EEPROM.h>                 // librairie pour lr stockage de données en mémoire EEPROM interne
#include <DHT.h>                       // librairie pour les sondes de températures DHT22
//#ifdef CHAUFCONF
    #include <PCF8574.h>            // librairie pour la gestion des PCF8574 I2c
//#endif

// ------ fichier de configuration du logiciel -------
#include "Config_DomoMaison.h"      // librairie contenant les variables spécifiques à chaque installation

// ------------------------------------------- Section Constantes et variables globales ---------------------------------
----------------------
// format des caractères de gestion de la trame Websocket (CSV)
#define STX '('                     // Caractère de dÃ©but de trame pour les messages
#define ETX ')'                     // Caractère de fin de trame pour les messages
#define Separator ';'               // Caractère Point-virgule séparateur pour les messages
// Longueur maximum de la trame utilisée par les WebSockets (ne pas modifier)..
#define MAX_FRAME_LENGTH 64
char EvtServer[126];                // chaine d'émission de message du serveur vers le client (126 caractÃ¨res maximum si non
extend)
char NumCard[] = NUM_BOARD;         // chaine représentant l'identifiant de la carte
byte PageWeb = 0;                           // numéro de la page Web en cours
// valeur courante du compteur pour le RTC
int Compteur_Rtc;
// valeur courante du compteur de pÃ©riode NTP
byte Compteur_Ntp;

//compteurs de ticks à la milliseconde pour synchronisation horloge
unsigned long Tick_Old = 0;
unsigned long Tick_New = 0;
unsigned long Epoch;                        // Heure Unix
unsigned long Epoch_Local;                  // Heure en local de la centrale

//compteur de tick pour l'envoi d'information entrées analogiques, compteurs d'impulsions et sondes température.
unsigned long Tick_Ana = 0;

//Indicateur de fonctionnement de la procédure d'envoi forcée des informations analogiques,des compteurs d'impulsion et des
sondes de température.
byte Indic_Init_Ana = 1;

// compteur de tick pour l'envoi d'information du compteur électrique
unsigned long Tick_Electrique = 0;

// indicateur de fonctionnement de la procédure d'envoi forcée des informations du compteur électrique
byte Indic_Init_Electrique = 1;

// Variables pour la gestion dt temps
byte Hours = 0;
byte Mins = 0;
byte Sec = 0;
byte DateJour = 0;
byte JourSemaine = 0;
byte Mois = 0;
int Annee = 0;

byte JourCourant = 0;                   // déclaration et initialisation de la variable jour de la semaine courant
byte Dls = 0;                           // Day Light Saving : heure d'hiver DLS = 0, d'heure d'été DLS = 1

int TempsMn = 0;                        // déclaration et initialisation variable temps courant en minutes
int TempsLDR = 0;                       // déclaration et initialisation variable temps détection pénombre LDR
int TempsLED = 0;                         // temps de clignotement LED
const int TEMPS_LED_NORMAL = 1000;      // durée du clignotement LED en ms (fonctionnement normal)
const int TEMPS_LED_PANNE = 200;        // durée du clignotement LED en ms (fonctionnement anomalie)

// Initialisation de l'afficheur LCD
const byte NUM_ROWS = 4;                                                        // nombre de lignes de l'afficheur
const byte NUM_COLS = 20;                                                       // nombre de colonnes de l'afficheur
LiquidCrystal lcd(LCD_RS, LCD_ENABLE, LCD_D4, LCD_D5, LCD_D6, LCD_D7);      // mode 4 bits - RW non connectée
int Compteur_Lcd;                                                                 //valeur courante du compteur de période
LCD.

// Variables pour le contrôle d'accès
const char CODE_CARTE[] = CODE_ACCES;
```

```
int Compteur_Deconnexion_Nb_Periode = 0;      // Compteur de période avant la déconnexion automatique
byte Indic_Acces = 0;                          // Indicateur de l'autorisation d'accès aux informations de la carte
byte Indic_Interdit = 0;                       // Indicateur d'émission de message d'interdiction d'accès

// Variables pour la carte SD
boolean CarteSD = false;
boolean TestSD = false;                        // variable mémorisation résultat renvoyé par fonction test (carte SD)

// Initialisation d'un serveur websocket
WebSocket wsServer("/",PORT_ARDUINO);          // répertoire racine dans la carte Arduino et port Ethernet

// Initialisation pour la partie dÃ©bogage
#ifdef DEBUG_MEM
    int Compteur_Test = 0;                    // pour TEST
#endif // DEBUG
// ****************************** Fin des déclaration pour le système de base
***********************************************************************

// ******************** Déclaration des variables pour la réception sans fil RFLink
*****************************************************************
char Buffin2[80] = "recep_rflink";      // Buffer de réception
int Bufflen2 = 0;                       // longueur du buffer
byte InByte2 = 0;                       // byte reçu sur la liaison série 2
// ******************** Fin des déclaration des variables pour la réception sans fil RFLink
*********************************************************

// ******************** Déclaration des variables pour la gestion des volets, du portail et des portes de garages
***********************************
byte NumProg[8];            // numéro de ligne du programme (correspond à un emplacement mémoire EEPROM
int HreOu[7];               // Heures d'ouverture des volets (en minutes)
int HreFe[7];               // Heures de fermeture des volets (en minutes)
byte NbProgV;               // Nombre de programmes pour la journée

int TempsApp = 500;         // temps d'appui sur les boutons de télécommande
// ****************** Fin des déclaration des variables pour la gestion des volets, du portail et des portes de garages
*****************************

// ******************** Déclaration des variables pour la gestion des temperatures filaires et sans fil
*********************************************
// le programme est prévu pour 12 sondes maximum (5 sondes filaires et 7 sondes radio)
// sondes filaires
int Compteur_Fil;                       // compteur pour la lecture des sondes filaires
int Compteur_CalculT;                   // compteur pour les calcul des minima et maxima des températures extérieure et
intérieure                                                                                          // Compteur pour la
lecture des sondes filaires
// Création des sondes de température filaires selon de nombre de sondes installées.
DHT dht1(PIN_DHT_1, DHT_TYPE);
DHT dht2(PIN_DHT_2, DHT_TYPE);
DHT dht3(PIN_DHT_3, DHT_TYPE);
DHT dht4(PIN_DHT_4, DHT_TYPE);
DHT dht5(PIN_DHT_5, DHT_TYPE);
DHT dht6(PIN_DHT_6, DHT_TYPE);

// sondes radio
byte IdT[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};                        // identifiant unique de la sonde
byte BattT[] = {10,10,10,10,10,10,10,10,10,10,10,10,10,10,10};          // niveau de la batterie de la sonde (10=faible, 90=OK)
byte EnregSonde = 0;                                                // drapeau indiquant si les sondes sont enregistrées en
Eprom (0 : pas des auvegarde effectuée)
// commun aux 2 types de sondes
// Les dernières données sont pour la sonde filaire derrière la centrale
int TempT[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};                      // température émise par la sonde (on divise ensuite par
10 pour avoir un décimal)
byte HumT[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};                     // humidité émise par la sonde

// Variables nécessaires pour la mémorisation des données des dernières 48 heures (visualisation sur graphe)
byte NumS_Ext;                  // numéro de la sonde appairée pour l'extérieur
byte NumS_Int;                  // numéro de la sonde appairée pour l'intérieur (1ère pièces)
int Temp_Ext[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int Temp_Int[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Températures mini et maxi extérieures et intérieures (les 2 premières températures dans la liste) pour enregistrement dans
SD Carte
int Temp_Ext_Mini = 1000;                         //valeur mini de la broche température extérieure pour la journée en cours.
int Temp_Ext_Maxi = -500;                         //valeur maxi de la broche température extérieure pour la journée en cours.
int Temp_Int_Mini = 1000;                         //valeur mini de la broche température intérieure pour la journée en cours.
int Temp_Int_Maxi = -500;                         //valeur maxi de la broche température intérieure pour la journée en cours.
// ******************** Fin des déclaration des variables pour la gestion des temperatures filaire et sans fil
**************************************

// ******************** Déclaration des variables pour la gestion de l'électricité et de la téléInformation
*****************************************

//---------------------------- Ajout pour téléinformation consommation EDF ---------------------------------------------------
-----------------------
char Buffin1[30] = "tele_cons";
int Bufflen1;                                  // Longueur d'un groupe téléinfo
char Adco1[13]= "000";                         //soit 12 + fin de chaine.
char Optarif1[5]= "****";                      //soit 4+ fin de chaine.
long Isousc1 = 0;
long Hchc1 = 0;
long Hchp1 = 0;
long Ejphn1 = 0;
long Ejphp1 = 0;
char Ptec1[5] = "****";                        //soit 4+ fin de chaine.
long Iinst1 = 0;
long Imax1 = 0;
long Papp1 = 0;
char Hhphc1 = '*';                               //soit 1 seul caractère.
char Motdetat1[7] = "000000";                   //soit 6+ fin de chaine.
byte InByte1 = 0;                              // byte reçu sur la liaison série du compteur électrique.
//
byte Indic_Teleinfo1 = 0;                      //Indicateur de fonctionnement de la téléinformation EDF. avec valeur 1=
détection OK.
```

```
unsigned long Compteur_Detection_Teleinfo1 = 0; //Compteur avant de déclarer la téléinformation hors service.
//
static char Deb = 0x0a;                          // caractère LF en hexadécimal
static char Fin = 0x0d;                          // caractère CR en héxadécimal

// heure courante
unsigned long Epoch_Teleinfo = 0;
//---------------------------- Fin Ajout pour téléinformation consommation EDF ----------------------------------------------
----------------------

//---------------------- Ajout mémorisation pour graphe sur 48 heures ------------------------------------------------------
-----------------------
byte IntHeure[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
//---compteur consommation
long Delta_IndexHC1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
long Delta_IndexHP1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
long IndexHC1 = 0;
long IndexHP1 = 0;
long Tempo_DeltaHC = 0;
long Tempo_DeltaHP = 0;

//indicateur de demande d'informations horaires pour graphe.
byte Indic_Info_Horaire = 0;
//---------------------- Fin Ajout mémorisation pour graphe sur 48 heures----------------------------------------------------
-----------------------

//---------------------- Ajout pour mémorisation dans SDCard de différentes informations -----------------------------------
-----------------------
// jour courant teleinfo
unsigned long Jour_Epoch_Teleinfo = 0;
// heure teleinfo
byte Heure_Epoch_Teleinfo;
byte Day_Jour_Teleinfo;

// jour précédent pour le stockage dans la SDcard
unsigned long Epoch_Veille = 0;

long Jour_IndexHC1 = 0;
long Jour_IndexHP1 = 0;

// valeur sur 9 caractères ascii d'un index ou un delta d'index
char Delta_Str[] = "000000000";

// Date Ascii----------------------
char Annee_Str[] = "0000";
char Mois_Str[] = "00";

char Nom_Fichier_Rep[30] = "    ";           //fin de chaine.
File LogFile;

//Enregistrement de log en ascii. avec Date, valeur consommation heures creuses, valeur consommation heures pleines,
//index heures creuses, index heures pleines, production, index production,
//température extérieure mini,température extérieure maxi,température intérieure mini,température intérieure maxi, compteur
eau
// les données production ne sont pas utilisées dans ce logciel, mais leur emplacement est prevu.
char Enr_Log[] = "0000-00-00;000000000;000000000;000000000;000000000;000000000;000000000;0000;0000;0000;0000;000000000  ";
int C_Enrlog = 0;     //caractère lu de la bibliothéque de la SDcard.

//indicateur de demande d'informations journalières de année/mois sur SDCard.
byte Indic_Info_Jour = 0;
//Année mois reçu de la demande.
char Annee_Recv[] = "0000";
char Mois_Recv[] = "00";
// ******************** Fin des déclaration des variables pour la gestion de l'électricité et de la téléInformation
*********************************

// ******************** Début des déclaration des variables pour la gestion du compteur d'eau
*****************************************************
// compteurs d'impulsions sous interruption
volatile unsigned long Compteur_Pulse_1 = 0;      // compteur d'eau sur broche 3
// ******************** Fin des déclaration des variables pour la gestion du compteur d'eau
*******************************************************

// ************** Début des déclaration des variables pour la gestion de l'alarme et des caméras
************************************************
//Indicateur présent/absent qui détermine si l'alarme est activée ou non
// si présent = 1 (alarme non activée) et absent = 0
// si alarme activée, envoi de mail sur détection des capteurs PIR, des contacteurs d'ouverture
byte Indic_Present_Absent = 1;                          // Indicateur de présence (Présent = 1, non présent = 0)
byte Indic_Journal_Alarme = 0;                          // Indicateur signalant l'envoi d'un journal d'alarme en cours
byte Indic_Effraction_Active = 0;                       // Indicateur signalant une effraction (Sirène et gyro actifs si pas
de mode Silence)
int TempsEffractionActive = 0;                              // Heure en minutes de la mise en marche de la sirène et du
gyrophare
byte Mode_Silence = 0;                                  // Indicateur si mode silence enclenché = 1 si activée(pas de sirène,
pas de gyro selon config, envoi mail)
// Variables pour l'activation de l'alarme
byte Demande_Active_Alarme = 0;                         // Demande d'activation de l'alarme
int TempsAlarmeActivation = 0;                          // Heures en minutes de la demande d'activation de l'alarme
// Variables pour la temporisation des déclenchement en zone 1 pour permettre l'arrêt de l'alarme
byte DeclenAlarme = 0;                                  // signale une effraction dans une zone temporisée (zone 1)
byte NumCapteur = 0;                                    // Numéro du capteur ayant eu un déclenchement dans une zone
temporisée (zone 1)
char TypeCapteur = 'x';                                 // Type de capteur ayant eu un déclenchement en zone temporisée
(Contacteur ou Détecteur IR)
int TempsAlarmeDeclen = 0;                              // Heures en minutes d'un déclenchement pour démarrage tempo
(seulement en zone 1)

// Variables pour les contacteurs d'ouverture et des détecteurs IR de l'alarme (10 contacteurs et 10 détecteurs
byte Mode_Contact[] = {1,1,1,1,1,1,1,1,1,1};            // Contacteurs ouverture activés
byte Mode_Detect[] = {1,1,1,1,1,1,1,1,1,1};             // détecteurs IR activés
byte Zone_Alarme[] = {1,1,1};                           // Zones de l'alarme, toutes les zones sont activées (3 zones :
Intérieur Maison, Extérieur Maison, Dépendances)
```

```
byte Indic_Alarme_Ouvert = 0;                           // indicateur de détection d'une ouverture
byte Indic_Alarme_PIR = 0;                              // indicateur de détection IR
byte Num_Indic_Con = 0;                                 // numéro du contacteur ouverture déclenché en dernier
byte Num_Indic_Pir = 0;                                 // numéro du détecteur IR déclenché en dernier
char Date_Dernier_Evt_Con[10][20] =
{"xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:
xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx"};
char Date_Dernier_Evt_Pir[10][20] =
{"xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:
xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx","xx/xx/xxxx;xx:xx:xx"};

// Variables pour les caméras
byte EtatIntrusion = 0;                                 // Indicateur de la détection Intrusion des caméras (0 = non activée,
1= activée)
byte EtatEnreg = 0;                                     // Indicateur de l'enregistrement programmé des caméras (0 = non
activé, 1= activé)
byte EtatPir = 0;                                       // Indicateur de l'enregistrement sur capteur PIR (0 = non activé, 1=
activé)
// ************** Fin des déclaration des variables pour la gestion de l'alarme et des caméras
************************************************

// ************** Début des déclaration des variables pour la gestion des fils pilote et du chauffage
*********************************************
#ifdef CHAUFCONF
    // Variables pour les fonctionnement des radiateurs
    boolean Ejp = false;
    boolean BascuEjp = false;                                       // pas d'EJP = 0, EJP = 1
    boolean Inib = false;                                           // permets d'hiniber le signal ejp et permettre le
fonctionnement des radiateurs (0=pas d'hinibion, 1=hinibition)
    byte ModeCours_Radia[] = {0,0,0,0,0,0,0,0};                     // mode de fonctionnement en cours des radiateurs des 8
zones de chauffage
    // 0=non connecté, 1=Confort, 2=Economique, 3=Hors gel, 4=Arrêt

    PCF8574 expander1;                                              // commande des fils pilote zones 1 à 4
    PCF8574 expander2;                                              // commande des fils pilote zones 5 à 8

    byte NumProgRad[8];                                             // numéro de la  programmation
    int HreProgRad[8];                                              // Heures d'application de la programmation (en minutes)
    byte NbreProgRad;                                               // Nombre de programmation pour la journée

#endif
// ************** Fin des déclaration des variables pour la gestion des fils pilote et du chauffage
*********************************************

// ************** Début des déclaration des variables pour la gestion d'envoi d'email
************************************************************
#ifdef MAILCONF
     char Expediteur_Smtp[] = LIB_EXPEDITEUR_MAIL;                  //expéditeur du mail.
     char Destinataire_Smtp[] = LIB_DESTINATAIRE_MAIL;              //destinataire du mail.
     char Copie_Smtp[] = LIB_COPIE_SMTP;                            //destinataire en copie.
     char Code_Auth_Plain[] = LIB_CODE_AUTH_PLAIN;                  //code authentification en mode PLAIN.
     //
     char Subject_Mail_Smtp[] = LIB_SUBJECT_MAIL;                   //texte du sujet du mail.
     char Message_Mail_Smtp[] = "                                ";  //information sur 30 caractères.
     char Date_Mail_Smtp[] = "Thu, 01 Jan 70 00:00:00 +0100 GMT";   //date heure du mail au format SMTP .

     byte Server_Mail[4];                                           //adresse courante du serveur mail.
     int Seq_Envoi_Mail = 0;                                        //sequenceur de l'automate d'envoi de mail.
     int Ksmtp = 0;                                                 //rang du serveur smtp utilisé.
     char SerInStringSmtp[50];                                      // array pour le stockage des caractères entrants par la
liaison Ethernet du serveur smtp.
     int Nbcarlu_Smtp = 0;
     unsigned int Tempo_Mail_Interval = 0;                          //temporisation d'envoi de mails.
     char Texte_Prov_Mail[12] = "";
     int Num_Message_Mail = 0;                                      //numéro du message de connexion à transmettre par mail.
     byte Indic_Start_Mail = 0;                                     //indicateur de démarrage de la carte pour la
transmission de mail.
     byte Indic_Mail_Controle_Journalier = 0;                       //indicateur de transmission du mail de controle
journalier.

     long Evenement_Pir = 0;                                        //date du dernier évenement de détection PIR,
     long Evenement_Con = 0;                                        //date du dernier évenement de contacteur ouverture,
     long Evenement_Papp = 0;                                       //date du dernier évenement de dépassement du seuil de
puissance en (VA).

     //déclaration du client de serveur internet du micro-ordinateur distant pour l'envoi des mails.
     EthernetClient client_mail;
#endif
// ************** Fin des déclaration des variables pour la gestion d'envoi d'email
************************************************************

// ----------------------------------------------------------------------------------------------------------------------------
-----------------------
// ********************************************** DEBUT DU SETUP
**************************************************************************
// ----------------------------------------------------------------------------------------------------------------------------
-----------------------
void setup()
{
     // ****************** PARTIE SYSTEME - INDISPENSABLE POUR LE BON FONCTIONNEMEN DU PROGRAMME
************************************************
     int i, j;                  // variable pour boucles

       #ifdef DEBUG
           Serial.begin(SPEED_TRANSMISSION);                                    // initialisation connexion Série pour
débogage
           Serial.println(F("Lancement du programme Centrale domotique"));
       #endif

   // ---------------------- Début de la configuration des broches Entrée/Sortie du système de base-------------------------
-----------------------
   pinMode(LCD_D7,OUTPUT);
   pinMode(LCD_D6,OUTPUT);
```

```
    pinMode(LCD_D5,OUTPUT);
    pinMode(LCD_D4,OUTPUT);
    pinMode(LCD_ENABLE,OUTPUT);
    pinMode(LCD_RS,OUTPUT);
    pinMode(PIN_BLLCD,OUTPUT);                          // utilisation de la broche analogique en broche digitale pour Back
Light LCD
    pinMode(PIN_LDR,INPUT);                             // broche de la luminosité de la LDR
    pinMode(PIN_LED_WATCHDOG,OUTPUT);                  // broche LED en sortie
    pinMode(PIN_WATCHDOG,OUTPUT);                           // met la broche watchdog en sortie

    pinMode(PIN_SELECT_SD,OUTPUT);                          // broche 4 pour l'Ethernet shield => CS SD
    pinMode(53,OUTPUT);                                     // laisser la broche SS en sortie - obligatoire avec librairie SD
    digitalWrite(53,HIGH);                                  // nÃ©cessaire pour SD avec Mega

    digitalWrite(PIN_BLLCD,1);                          // Back Light  Allumage de l'écran LCD
    TempsLED = TEMPS_LED_NORMAL;                        // fonctionnement normal et clignotement normal de la LED
    // ---------------------------- Fin de la configuration des broches Entrée/Sortie du programme système --------------------
-----------------------

    // --------------- Début de la configuration des broches Entrée/Sortie de la gestion volets, portail, porte de garage ----
-----------------------
    pinMode(PIN_VOL_GAUCHE,OUTPUT);
    pinMode(PIN_VOL_MILIEU,OUTPUT);
    pinMode(PIN_VOL_DROIT,OUTPUT);
    pinMode(PIN_VOL_HAUT,OUTPUT);
    pinMode(PIN_VOL_STOP,OUTPUT);
    pinMode(PIN_VOL_BAS,OUTPUT);
    pinMode(PIN_PORTAIL,OUTPUT);
    pinMode(PIN_LIBRE_1,OUTPUT);
    pinMode(PIN_GARAGE_1,OUTPUT);
    pinMode(PIN_GARAGE_2,OUTPUT);
    // --------------- Fin de la configuration des broches Entrée/Sortie de la gestion volets, portail, porte de garage ------
-----------------------

    // --------------- Début de la configuration des broches Entrée/Sortie de la gestion des températures --------------------
-----------------------
    pinMode(PIN_DHT_1,INPUT);
    pinMode(PIN_DHT_2,INPUT);
    pinMode(PIN_DHT_3,INPUT);
    pinMode(PIN_DHT_4,INPUT);
    pinMode(PIN_DHT_5,INPUT);
    // --------------- Fin de la configuration des broches Entrée/Sortie de la gestion des températures ----------------------
-----------------------

    // --------------- Début de la configuration des broches Entrée/Sortie de la gestion de l'électricité --------------------
-----------------------
    pinMode(PIN_EJP, INPUT_PULLUP);                 // mise en entrée de la broche avec résistance pull-up pour mise en niveau
haut
    // --------------- Fin de la configuration des broches Entrée/Sortie de la gestion de l'électricité ----------------------
-----------------------

    // --------------- Début de la configuration des broches Entrée/Sortie de la gestion de l'alarme ----------------------
-----------------------
    pinMode(PIN_SIRENE,OUTPUT);                 // Broche de commande de la sirène
    pinMode(PIN_GYRO,OUTPUT);                   // Broche de commande du gyrophare
    // Mise au niveau haut des broches de commande du gyrophare et de la sirène (la commande se fait par la mise en niveau
bas)
    digitalWrite(PIN_SIRENE,HIGH);
    digitalWrite(PIN_GYRO,HIGH);
    // --------------- Fin de la configuration des broches Entrée/Sortie de la gestion de l'électrcité ----------------------
-----------------------

    // ---------------------------- Début de la partie système de base du programme ----------------------------------------
-----------------------
    // Initilisation de l'afficheur LCD
    lcd.begin(NUM_COLS,NUM_ROWS);                       // afficheur 20 colonnes, 4 lignes
    lcd.clear();                                        // efface LCD
    lcd.setCursor(0,0);                                 // positionne le curseur colonne 1, ligne 1
    lcd.print("DOMOMAISON 2017");
    lcd.setCursor(0,1);
    lcd.print("Version 1.70");
    delay(1000);

    // initialisation du compteur pour le rafraichissement de l'afficheur
    Compteur_Lcd = 0;

    // Initialisation de la carte SD (à supprimer si vous ne mettez pas de carte)
    testCarteSD();                                      // Test de la présence d'une carte SD et de son bon fonctionnement

    // Initialisation des compteur RTC et NTP
     Compteur_Rtc = 0;
     Compteur_Ntp = 0;

     // Initialisation horloge interne de la carte
    Tick_Old = 0L;
    Tick_New = 0L;

    lectureRTC();               // Lecture de la date et de l'heure sur le module DS1307RTC et initialisation des variables
Temps

    // Mise à jour de l'horloge du DS1307 à partir d'un site NTP
        if (MISE_JOUR_NTP == 1) {                       // Mode mise à jour horloge par NTP demandée
            lcd.setCursor(8,3);                     // positionne curseur colonne 9, ligne 4
            lcd.print("NTP en cours");
                #ifdef DEBUG
                    Serial.println(F("NTP en cours"));
                #endif
            Ethernet.begin(MAC);
                if (Ethernet.begin(MAC) == 0) {         // Si la connexion web échoue
                    lcd.setCursor(8,3);                  // positionne curseur colonne 9, ligne 4
                    lcd.print("Pb NTP      ");
                }
                else
```

```
                {
                    calcul_ntp_time(ntp());            // on s'occupe d'abord de récupérer l'heure via serveur NTP +
affichage
                    lectureRTC();                      // on relie le RTC pour le calcul des dates et heures
                }
        }

    JourCourant = JourSemaine;                         // mémorisation du jour de la semaine en cours
    TempsMn = Hours * 60 + Mins;                       // calcul du temps courant en minutes

     affiDateHeure(2);                                  // affichage de la date et de l'heure en ligne 3

     // Initialisation de la carte Ethernet
    Ethernet.begin(MAC, IP_LOCAL);                     // avec ip fixe
    //Ethernet.begin(MAC);                             // avec ip DHCP
    delay(1000);
        #ifdef DEBUG
            Serial.print(F("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " ));
            Serial.println(Ethernet.localIP());
        #endif
    wsServer.registerConnectCallback(&onConnect);
    wsServer.registerDataCallback(&onData);
    wsServer.registerDisconnectCallback(&onDisconnect);
    wsServer.begin();
        #ifdef DEBUG
            Serial.println(F("Serveur Ethernet OK : Ecoute sur port 80 (http)"));
        #endif

// Initialisation des variables d'accÃ¨s
    Indic_Acces = 0;                                   // Indicateur d'accès à la carte
    Indic_Interdit = 0;                                // Indicateur d'émission de message d'interdiction

     delay(3000);                                             // attente de 3 secondes pour permettre la lecture de
l'affichage

    lcd.clear();                                        // efface LCD
    lcd.setCursor(8, 3);                               // positionne curseur colonne 9, ligne 3

           if (MISE_JOUR_NTP == 1) {
                 lcd.print("Avec NTP   ");
           }
           else {
                 lcd.print("Sans NTP   ");
           }
    // ---------------------------- Fin de la partie système de base du programme ----------------------------------------
----------------------

    // ---------------------- Début de la partie réception sans fil module RFLink ----------------------------------------
----------------------
     // Initialisation communication RFLink
    Serial2.begin(57600);
    // ---------------------- Fin de la partie réception sans fil module RFLink ----------------------------------------
---------------------

    // ---------------------- Début de la partie gestion des volets, portail et portes de garage ----------------------------
----------------------
    // Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
    // Initialisation du temps de fonctionnement des volets (limité à 20 volets)
    /* EEPROM.write(0,1);
    EEPROM.write(1,1);
        for (int i=2; i <= 19; i++){
            EEPROM.write(i,10);
        } */

    // Initialisation de la programmation des volets
        // 1000 à 1023 = prog. 1, 1024 à 1047 = prog. 2, 1048 à 1071 = prog. 3, 1072 à 1095 = prog. 4
        // 1096 à 1119 = prog. 5, 1120 à 1143 = prog. 6, 1144 à 1167 = prog. 7, 1168 à 1191 = prog. 8
        // 1192 à 1399 = réservé pour la possibilité de passer à 16 programmes.
        /* for (i=0; i<4; i++) {
            EEPROM.write(1000 + (i * 24),0);
            EEPROM.write(1001 + (i * 24),32);       // 32 pour semeine, 33 pour week-end
            EEPROM.write(1002 + (i * 24),32);       // 32 pour semeine, 33 pour week-end
            EEPROM.write(1003 + (i * 24),32);       // 32 pour semeine, 33 pour week-end
                for (j=0; j<16; j++) {
                    EEPROM.write(1004 + (i * 24) + j,0);
                }
            EEPROM.write(1020 + (i * 24),99);       // heure ouverture = 0 à 23, 99 pour aucune heure
            EEPROM.write(1021 + (i * 24),99);       // minute ouverture = 0 à 59, 99 pour aucune minute
            EEPROM.write(1022 + (i * 24),99);       // heure fermeture = 0 à 23, 99 pour aucune heure
            EEPROM.write(1023 + (i * 24),99);       // minute fermeture = 0 à 23, 99 pour aucune minute
        }
        for (i=4; i<8; i++) {
            EEPROM.write(1000 + (i * 24),0);
            EEPROM.write(1001 + (i * 24),99);       // 99 pour pas de date sélectionnée
            EEPROM.write(1002 + (i * 24),99);
            EEPROM.write(1003 + (i * 24),99);
                for (j=0; j<16; j++) {
                    EEPROM.write(1004 + (i * 24) + j,0);
                }
            EEPROM.write(1020 + (i * 24),99);       // heure ouverture = 0 à 23, 99 pour aucune heure
            EEPROM.write(1021 + (i * 24),99);       // minute ouverture = 0 à 59, 99 pour aucune minute
            EEPROM.write(1022 + (i * 24),99);       // heure fermeture = 0 à 23, 99 pour aucune heure
            EEPROM.write(1023 + (i * 24),99);       // minute fermeture = 0 à 23, 99 pour aucune minute
        } */

    // Vérifie s'il ya une programmation pour les volets
    charger_prog_volet();                                          // chargement de la programmation des volets pour la journée
    // ---------------------- Fin de la partie gestion des volets, portail et portes de garage -----------------------------
----------------------

    // ---------------------- Début de la partie Gestion des températures ------------------------------------------------
----------------------
```

```
    // Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
    // Initialisation de l'appairage des sondes de température
    //    for (i=20; i <= 34; i++){
    //        EEPROM.write(i,0);
    //    }

    // Initialisation de l'ordre des sondes de température sans fil reçues par la centrale domotique
    //    for (i=35; i <= 45; i++){
    //        EEPROM.write(i,0);
    //   }

     // Démarrage des sondes filaires
     Compteur_Fil = 0;                          // initialisation du compteur pour la lecture des sondes filaires

       if (NB_SONDEST_F > 0) {
           dht1.begin();
       }
       if (NB_SONDEST_F > 1) {
           dht2.begin();
       }
       if (NB_SONDEST_F > 2) {
           dht3.begin();
       }
       if (NB_SONDEST_F > 3) {
           dht4.begin();
       }
       if (NB_SONDEST_F > 4) {
           dht5.begin();
       }
    dht6.begin();                   // sonde montée derrière la centrale domotique
     //
     // Chargement de l'ordre des sondes de température sans fil
     EnregSonde = EEPROM.read(45);
       if (EnregSonde == 1) {                     // Ordre enregistré
           for (i=0; i<10; i++) {
               IdT[i + 5]=EEPROM.read(35 + i);     // le numéro des sondes sans fil commence à 5 (0 à 4 = sondes filaires)
           }
       }
    // initialisation du compteur pour calcul T° sondes
    Compteur_CalculT = 0;
    // Chargement du numéro des sondes appairées pour l'extérieur et l'intérieur
    NumS_Ext = EEPROM.read(20) - 1;     // -1 car l'Id, laT° et l'humidité des sondes démarre à 0 en indice dans les tableaux
    NumS_Int =  EEPROM.read(21) - 1;    // -1 car l'Id, laT° et l'humidité des sondes démarre à 0 en indice dans les tableaux
    // ---------------------- Fin de la partie Gestion des températures ----------------------------------------------------
----------------------

    // ---------------------- Début de la partie Gestion de l'électrcité et de la téléInformation --------------------------
----------------------
    //
    // Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
    // Initialisation des compteurs et des objectifs électriques, concerne également le compteur Eau et l'objectif
    //    sauverInt(100, 2016);            // Année de la dernière sauvegarde
    //    for (i=102; i < 117; i++){
    //        EEPROM.write(i,0);
    //    }

    // Ajout pour teleinfo compteur de consommation
    Serial1.begin(1200);            //Vitesse imposé par le compteur électrique EDF.
    // parite paire E
    // 7 bits data
    UCSR1C = B00100100;                    // passage en mode 7bits + bit de parité de la liaison série pour la liaison avec
le compteur EDF.
        #ifdef DEBUG_ELEC
            Serial.println(F("initialisation TELEINFO consommation"));
        #endif
    Indic_Teleinfo1 = 0;
    Compteur_Detection_Teleinfo1 = 0;       //réarmement du compteur de détection téléinfo.
    Indic_Info_Horaire = 0;                 //initialisation de l'indicateur de demande d'informations horaires
    Indic_Info_Jour = 0;                    //initialisation de l'indicateur de demande d'informations journalière
    Epoch_Teleinfo = 0;
    // Init pour informations journalières
    Jour_Epoch_Teleinfo = 0;
    Day_Jour_Teleinfo = 0;
    Heure_Epoch_Teleinfo = 0;
    // ---------------------- Fin de la partie Gestion de l'électricité et de la téléInformation --------------------------
----------------------

    // ---------------------------------- Début de la partie de gestion de l'eau -----------------------------------------
----------------------
    // ---------------------------------- Fin de la partie de gestion de l'eau -------------------------------------------
----------------------

    // ---------------------- Début de la partie Gestion de l'alarme et des caméras ----------------------------------------
----------------------
    // Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
    // Initialisation des contacteurs d'ouverture et des détecteurs IR (actvés)
    //    for (i=46; i < 66; i++){
    //        EEPROM.write(i,1);
    //   }
    // Initialisation des états de fonctionnement des caméras (Détection Intrusion, Enregistrement programmé, Enregistrement
sur capteur PIR)
        for (i=66; i<69; i++) {
            EEPROM.write(i,0);                          // Mode non activé)
        }

        if (MODE_FONCT_ALARME == "ON") {
            Demande_Active_Alarme = 1;                  // Demande d'activation de l'alarme
            TempsAlarmeActivation = TempsMn;            // Initialisation de l'heure de la demande pour temporisation
        }
    Indic_Present_Absent = 1;                           // Alarme pas encore active, attente de la tempo
```

```
    Indic_Journal_Alarme = 0;

       // Chargement du mode des contacteurs et des détecteurs (actif=1, inactif=0)
        for (i=0; i < 10; i++){
            Mode_Contact[i] = EEPROM.read(46 + i);
       }
        for (i=0; i < 10; i++){
            Mode_Detect[i] = EEPROM.read(56 + i);
       }
    // ---------------------- Fin de la partie Gestion de l'alarme et des caméras ----------------------------------------
----------------------

    // ---------------------- Début de la partie Gestion des fils pilotes et du chauffage ---------------------------------
----------------------
    #ifdef CHAUFCONF
        // Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire
lors du premier démarrage)
        // Initialisation  du mode de fonctionnement immédiat des radiateurs (en cours et précédent)
            // for (i=80; i<96; i++) {
            //    EEPROM.write(i,1);          // mode Confort par défaut
            // }

        // Initialisation de la programmation des radiateurs
// 1400 à 1421 = prog. 1, 1422 à 1443 = prog. 2, 1444 à 1465 = prog. 3, 1466 à 1487 = prog. 4
// 1488 à 1509 = prog. 5, 1510 à 1531 = prog. 6, 1532 à 1553 = prog. 7, 1554 à 1575 = prog. 8
// 1576 à 1799 = réservé pour la possibilité de passer à 16 programmes.
    // for (i=0; i<4; i++) {
    //    EEPROM.write(1400 + (i * 22),0);
    //    EEPROM.write(1401 + (i * 22),32);       // 32 pour semeine, 33 pour week-end
    //    EEPROM.write(1402 + (i * 22),32);       // 32 pour semeine, 33 pour week-end
    //    EEPROM.write(1403 + (i * 22),32);       // 32 pour semeine, 33 pour week-end
    //        for (j=0; j<16; j++) {
    //            EEPROM.write(1404 + (i * 22) + j,0);
    //        }
    //    EEPROM.write(1420 + (i * 22),99);       // heure ouverture = 0 à 23, 99 pour aucune heure
    //    EEPROM.write(1421 + (i * 22),99);       // minute ouverture = 0 à 59, 99 pour aucune minute
    // }
    // for (i=4; i<8; i++) {
    //    EEPROM.write(1400 + (i * 22),0);
    //    EEPROM.write(1401 + (i * 22),99);       // 99 pour pas de date sélectionnée
    //    EEPROM.write(1402 + (i * 22),99);
    //    EEPROM.write(1403 + (i * 22),99);
    //        for (j=0; j<16; j++) {
    //            EEPROM.write(1404 + (i * 22) + j,0);
    //        }
    //    EEPROM.write(1420 + (i * 22),99);       // heure ouverture = 0 à 23, 99 pour aucune heure
    //    EEPROM.write(1421 + (i * 22),99);       // minute ouverture = 0 à 59, 99 pour aucune minute
    // }
// Initialisation pour les fils pilote
    expander1.begin(0x38);
    expander2.begin(0x39);
        for (i=0; i <8; i++){
                    expander1.pinMode(i, OUTPUT);
                    expander1.digitalWrite(i, HIGH);        // mise en niveau haut des sorties (pas de 220V sur les fils
pilote)
                    expander2.pinMode(i, OUTPUT);
                    expander2.digitalWrite(i, HIGH);        // mise en niveau haut des sorties (pas de 220V sur les fils
pilote)
                }

        // charge et applique le mode de fonctionnement immédiat
                for (i=0; i <8; i++){
                    ModeCours_Radia[i] = EEPROM.read(80 + i);
                }
            commande_immediat_radia();                      // commande les radiateurs en mode par défaut
            charger_prog_radia();                           // Vérifie et charge les programmations du jour
    #endif
    // ---------------------- Fin de la partie Gestion des fils pilotes et du chauffage ----------------------------------
----------------------

    // ---------------------- Début de la partie Gestion de l'envoi d'email ------------------------------------------------
----------------------
     #ifdef MAILCONF
          Nbcarlu_Smtp = sizeof(SerInStringSmtp);
          Seq_Envoi_Mail = 0;                               //séquenceur de l'automate d'envoi de message.
          Ksmtp = 0;                                        //numero de séquence interne au test de connexion au serveur
smtp correspondant au rang du serveur par priorité.
          Tempo_Mail_Interval = 0;                          //compteur de temps de l'automate d'envoi (fait office de delay
sans bloquer la carte).
          Num_Message_Mail = 0;                             //numero courant du message à transmettre par mail.
          Indic_Start_Mail = 0;                             //indicateur de démarrage de la carte.
          Indic_Mail_Controle_Journalier = 0;               //indicateur de transmission du mail de controle journalier.
          #ifdef DEBUG_MAIL
               Serial.println(F("lancement Gestion e-mail" ));
          #endif
     #endif
    // ---------------------- Fin de la partie Gestion de l'envoi d'email ------------------------------------------------
----------------------
}
// ------------------------------------------------------------------------------------------------------------------------
----------------------
// ************************************************ FIN DU SETUP
**********************************************************************
// ------------------------------------------------------------------------------------------------------------------------
----------------------


// ------------------------------------------------------------------------------------------------------------------------
----------------------
// ************************************************ DEBUT DU LOOP
**********************************************************************
// ------------------------------------------------------------------------------------------------------------------------
----------------------
```

```
void loop()
{
    // ------------------- Début Partie de base du programme système ----------------------------------------------------------
-----------------------
    int i, j;

    watchdog();                                                         // appel de la fonction reset Watchdog

           if(MISE_JOUR_NTP == 1 && Compteur_Ntp == CYCLE_NTP) {         // mise à jour horloge par NTP
                calcul_ntp_time(ntp());
           }


        if (Compteur_Rtc == 0) {
            lectureRTC();                                       // appel de la fonction lecture données RTC DS1307
        }
      Compteur_Rtc = Compteur_Rtc + 1;
        if (Compteur_Rtc >= CYCLE_RTC) Compteur_Rtc = 0;// Reset compteur de boucle de demande auprès de l'horloge RTC DS1307

           if (JourSemaine != JourCourant) {                    // test si changement de jour
            JourCourant = JourSemaine;                          // mémorisation du jour de la semaine
            Compteur_Ntp = Compteur_Ntp + 1;                    // incrémentation tous les jours du compteur pour mise à jour de
l'heure par un site NTP
            // avec module DM_volets.ino
            charger_prog_volet();
            // avec module DM_chauffage
            #ifdef CHAUFCONF
                charger_prog_radia();                           // chargement de la programmation des radiateurs pour la journée
            #endif
           }

    TempsMn = Hours * 60 + Mins;                                // calcul du temps courant en minutes
    Tick_New = millis();                                        // Sauvegarde du tick courant

        if (Compteur_Lcd == 0){
            affiDateHeure(0);                                   // appel de la fonction affichage date et heure dans la boucle
loop
            affichageMode();                                    // appel de la fonction affichage du mode de fonctionnement sur
LCD ligne 2
        }

    Compteur_Lcd = Compteur_Lcd + 1;

           if (Compteur_Lcd >= CYCLE_LCD) Compteur_Lcd = 0;    //Reset compteur de boucle pour affichage.

    testLDR();                                                  // appel de la fonction test de la luminosité => extinction de
Back Light LCD dans la pénombre

    blinkLED(TempsLED);                                         // appel de la fonction clignotement de la LED en Face Avant

    wsServer.listen();                                          // Ecoute d'informations sur le port Ethernet de la WebSocket

        if (wsServer.isConnected() && (Indic_Acces == 2)) {
            Indic_Acces = 1;                                      // L'échéance d'interdiction n'a pas été atteint
        }

        if ((wsServer.isConnected()) && (Indic_Acces == 1)) {
            // envoi des données vers le serveur web
            // Envoi des valeurs des entrées analogiques, des compteurs d'impulsions et des sondes de température, de l'alarme
vers le client WebSocket.
            if (Indic_Init_Ana == 1){
                if ((PageWeb == 1) || (PageWeb == 11)) {        // page acceuil et paramètres
                    envoi_web_sondesT();                          // envoi des sondes de température
                    Indic_Init_Ana = 0;                           // Fin de la procédure d'envoi des informations analogiques,
des compteurs d'impulsions et des sondes de températures.
                }
                if (PageWeb == 5) {                             // page Chauffage
                    envoi_web_sondesT();                          // envoi des sondes de température
                    #ifdef CHAUFCONF
                        envoi_mode_radia();                     // envoi du mode de fonctionnement des radiateurs
                    #endif
                    Indic_Init_Ana = 0;                           // Fin de la procédure d'envoi des informations analogiques,
des compteurs d'impulsions et des sondes de températures.
                }
                if (PageWeb == 7) {                             // page Protection
                    envoi_web_alarme();                         // envoi des informations de l'alarme (activation,
déclenchements)
                    Indic_Init_Ana = 0;                           // Fin de la procédure d'envoi des informations analogiques,
des compteurs d'impulsions et des sondes de températures.
                }
            }
            // Envoi des informations des compteurs électriques.
            if (Indic_Init_Electrique == 1) {
                      if (PageWeb == 1) {
                    Indic_Init_Electrique = 0;
                    envoi_info_elect();                                 // module DM_electricte : envoi de l'information
téléinfo de la puissance instantannée et lesignal ejp (Page web 1)
                      }
                      else if (PageWeb == 8){
                    Indic_Init_Electrique = 0;
                    envoi_web_teleinfo();                       // module DM_electricite : Envoi des informations téléinfo du
compteur électrique consommation à l'interface Web
                      }
            }
             Compteur_Deconnexion_Nb_Periode = 0;
        }
        else {
            if ((!wsServer.isConnected()) && (Indic_Acces == 1)) {
                // Armement d'une temporisation avant d'interdire l'accÃ¨s car connexion WebSocket fermée
                Compteur_Deconnexion_Nb_Periode = NB_PERIODE_DECONNEXION;
                Indic_Acces = 2;
                // ré-armement de l'indicateur d'envoi des informations du compteur électrique pour la prochaine connexion par
websocket
```

```
                Indic_Init_Electrique = 1;
            }
        }

      // Controle de l'échéance de l'autorisation d'accès
        if ((Indic_Acces == 2) && (Compteur_Deconnexion_Nb_Periode >= 0)){
            Compteur_Deconnexion_Nb_Periode --;                                  //décrementation de l'échéance.
            if (Compteur_Deconnexion_Nb_Periode == 0){
                Indic_Acces = 0;                                                 //interdiction d'accès.
                #ifdef DEBUG
                    Serial.println(F("interdiction d'acces"));
                #endif
            }
        }

        //Réponse au client en cas de demande alors qu'il y a interdiction d'accès.
        if (wsServer.isConnected() && (Indic_Interdit == 1)){
            // rien pour l'instant
        }

        // Incrémentation et reset du compteur de cycle d'envoi des entrées analogiques,des compteurs d'impulsions et des
sondes de température.
        if ((Tick_New >= Tick_Ana) && ((Tick_New - Tick_Ana) >= PERIODE_CYCLE_ANA) && (Indic_Init_Ana == 0)) {
            Indic_Init_Ana = 1;     //Equivaut à autoriser un nouvel envoi des entrées analogiques et des compteurs
d'impulsions par le serveur.
            Tick_Ana = Tick_New;
        }
        else if ((Tick_New < Tick_Ana) && ((ULONG_MAX - Tick_Ana) + Tick_New >= PERIODE_CYCLE_ANA) && (Indic_Init_Ana == 0)){
            Indic_Init_Ana = 1;     //Equivaut à autoriser un nouvel envoi des entrées analogiques et des compteurs
d'impulsions par le serveur.
            Tick_Ana = Tick_New;
        }

        // Incrémentation et reset du compteur de cycle d'envoi des compteurs électriques.
        if ((Tick_New >= Tick_Electrique) && ((Tick_New - Tick_Electrique) >= PERIODE_CYCLE_ELECTRIQUE) &&
(Indic_Init_Electrique == 0)) {
            Indic_Init_Electrique = 1;      //Equivaut à autoriser un nouvel envoi des compteurs électriques..
            Tick_Electrique = Tick_New;
        }
        else if ((Tick_New < Tick_Electrique) && ((ULONG_MAX - Tick_Electrique) + Tick_New >= PERIODE_CYCLE_ELECTRIQUE) &&
(Indic_Init_Electrique == 0)){
            Indic_Init_Electrique = 1;      //Equivaut à autoriser un nouvel envoi des compteurs électriques.
            Tick_Electrique = Tick_New;
        }
    // -------------------------- Fin de la partie de base du programme système ------------------------------------------
----------------------

    // --------------------- Début Partie réception module RFLink ---------------------------------------------------------
----------------------
    read_reception_rflink();                        // Lecture du module RFLink (433 Mhz et 2,4 Ghz
    // --------------------- Fin de la partie réception module RFLink -----------------------------------------------------
----------------------

    // --------------------- Début Partie Gestion des volets, portail, portes de garage -----------------------------------
----------------------
    applique_prog_volets();                     // Fonction appliquant la programmation des volets s'il y en a une
    // -------------------- Fin de la partie Gestion des volets, portail, portes de garage -----------------------------------
----------------------

    // --------------------- Début Partie gestion des températures ---------------------------------------------------------
----------------------
     // Lecture des sondes filaires
          if (Compteur_Fil == 0) {                                              // lecture des sondes filaires à intervalle
programmé
           if(NB_SONDEST_F > 0) {
               TempT[0] = dht1.readTemperature() * 10;          // lecture température en °C sonde 1
               HumT[0] = dht1.readHumidity();                   // lecture humidité sonde 1
           }
           if(NB_SONDEST_F > 1) {
               TempT[1] = dht2.readTemperature() * 10;          // lecture température en °C sonde 2
               HumT[1] = dht2.readHumidity();                   // lecture humidité sonde 2
           }
           if(NB_SONDEST_F > 2) {
               TempT[2] = dht3.readTemperature() * 10;          // lecture température en °C sonde 3
               HumT[2] = dht3.readHumidity();                   // lecture humidité sonde 3
           }
           if(NB_SONDEST_F > 3) {
               TempT[3] = dht4.readTemperature() * 10;          // lecture température en °C sonde 4
               HumT[3] = dht4.readHumidity();                   // lecture humidité sonde 4
           }
           if(NB_SONDEST_F > 4) {
               TempT[4] = dht5.readTemperature() * 10;          // lecture température en °C sonde 5
               HumT[4] = dht5.readHumidity();                   // lecture humidité sonde 5
           }
           TempT[15] = dht6.readTemperature() * 10;             // lecture température en °C sonde 6 (Sonde montée arrière
boitier centrale)
          }
    Compteur_Fil = Compteur_Fil + 1;
        if (Compteur_Fil >= CYCLE_FIL) Compteur_Fil = 0;          // lit les condes filaires à un intervalles programmé
    // fin des lecture filaire

     // Traitement des températures minima et maxima.
          if (Compteur_CalculT == CYCLE_CALCUL_TE) {                        // calcul des températures extremes à intervalle
programmé
           if (int(TempT[NumS_Int] != 0)) {
               if (int(TempT[NumS_Int]) > Temp_Int_Maxi) Temp_Int_Maxi = int(TempT[NumS_Int]);
               if (int(TempT[NumS_Int]) < Temp_Int_Mini) Temp_Int_Mini = int(TempT[NumS_Int]);
           }
           if (int(TempT[NumS_Ext] != 0)) {
               if (int(TempT[NumS_Ext]) > Temp_Ext_Maxi) Temp_Ext_Maxi = int(TempT[NumS_Ext]);
               if (int(TempT[NumS_Ext]) < Temp_Ext_Mini) Temp_Ext_Mini = int(TempT[NumS_Ext]);
           }
          }
```

```
            //
    Compteur_CalculT = Compteur_CalculT + 1;
        if (Compteur_CalculT > CYCLE_CALCUL_TE) Compteur_CalculT = 0;   // calcule les extremes des température à un
intervalles programmé
    // ---------------------- Fin Partie gestion des températures ------------------------------------------------------------
----------------------

    // ---------------------- Début Partie gestion de l'électricité et de la téléInformation ---------------------------------
----------------------
    read_teleinfo_consommation();

    Compteur_Detection_Teleinfo1 = Compteur_Detection_Teleinfo1 + 1;
    //controle du fonctionnement de la téléinformation avec indicateur à  0 si dépassement du temps max entre deux groupes en
réception .
        if (Compteur_Detection_Teleinfo1 > DETECTION_TELEINFO){
            Indic_Teleinfo1 = 0;            //téléinformation hors service
                #ifdef DEBUG_ELEC
                    Serial.println(F("Téléinformation hors service"));
                #endif      // DEBUG_TINFO
            Compteur_Detection_Teleinfo1--;
        }

     // Traitement index de consommation horaire pour les compteurs consommation et production sur les 48 heures précédentes
consécutives pour graphe
      if ((Hours != Heure_Epoch_Teleinfo) && (millis() > 4000) && (Annee != 1970)){
        mise_jour_teleinfo_h48();
      }
      else if (millis()>3000){
           //calcul des différences de façon permanente pour affichage des informations de l'heure courante.
        calcul_diff_affichage_graphe();
    }
    // ---------------------- Fin Partie gestion de l'électricité et de la téléInformation ---------------------------------
----------------------

    // ---------------------- Début Partie Enregistrement dans SD Carte ------------------------------------------------------
----------------------
      //----------------Traitement index de consommation journalière avec enregistrement dans SDcard--------------------------
---
      if ((DateJour != Day_Jour_Teleinfo) && (millis() > 4000) && (Epoch_Local > Jour_Epoch_Teleinfo) && (Annee!= 1970)){
        enreg_infoJour_SD();
      }
    // Fonction dans module DM_electricite
    // ---------------------- Fin Partie Enregistrement dans SD Carte ------------------------------------------------------
----------------------

    // ---------------------- Début Partie Gestion de l'alarme ---------------------------------------------------------------
----------------------
        // Activation de l'alarme
        if (Demande_Active_Alarme == 1) {                                       // Demande d'activation de l'alarme
            if ((TempsMn - TempsAlarmeActivation) >= TEMPO_MARCHE_ALARME) {
                Indic_Present_Absent = 0;                                       // Activation de l'alarme après temporisation
                Demande_Active_Alarme = 0;
                envoi_etatAlarme();                                             // Envoi au clavier déporté le nouvel état de
l'alarme
            }
        }

        // Déclenchement temporisé
        if (DeclenAlarme == 1) {                                            // un déclenchement temporisé alarme est mémorisé
            if ((TempsMn - TempsAlarmeDeclen) >= TEMPO_VALID_ALARME) {
                if (TypeCapteur == 'C') {
                    declenchement_contacteur(NumCapteur);
                }
                else if(TypeCapteur == 'D') {
                    declenchement_detecteur(NumCapteur);
                }
                DeclenAlarme = 0;
                TempsAlarmeDeclen = 0;
                }
        }

        // Signalement effraction en cours (calcul le temps pour la coupure) - (Sirène et gyro en marche si pas de mode
Silence)
        if (Indic_Effraction_Active == 1) {         // la sirène est en marche
            if ((TempsMn - TempsEffractionActive) >= DUREE_EFFRACTION_ALARME) {
                Indic_Effraction_Active = 0;                                    // indique la coupure de la sirène et du gyro
s'ils sont actifs
                coupe_gyro_sirene();                                        // arrête le gyrophare et la sirène (Module
DM_protection)
            }
        }
    // ---------------------- Fin Partie Gestion de l'alarme -----------------------------------------------------------------
----------------------

    // ---------------------- Début Partie Gestion des fil pilote et du chauffage --------------------------------------------
----------------------
    #ifdef CHAUFCONF
        // Vérifie la présence du signal EJP. En cas d'EJP, arrêt des radiateurs (si le signal d'hinition n'est pas actif)
        Ejp = !digitalRead(PIN_EJP);                    // lecture du signal EJP

            if (Ejp != BascuEjp) {                      // Changement d'état du signa EJP
                if (Ejp == true && Inib == false) {
                    commandeEjp_Rad();                  // Arrêt des radiateurs des 8 zones
                }
                else {
                    commande_immediat_radia();          // Retour au mode de fonctionnement précédent des radiateurs
                    BascuEjp = Ejp;                     // mise en mémoire de l'état Ejp
                }
            }
        if (!Ejp) {
            applique_prog_radia();                      // appel de la fonction d'application de la programmation des
radiateurs
        }
```

```
    #endif
    // ---------------------- Fin Partie Gestion des fil pilote et du chauffage ----------------------------------------------
-----------------------

     //-------------Début du programme automate envoi de e-mail avec XX secondes  d'attente avant de tester un autre serveur
smtp --------------------
     #ifdef MAILCONF
       verifie_envoi_email();
    #endif
     //-------------Fin du programme automate envoi de e-mail avec XX secondes  d'attente avant de tester un autre serveur
smtp ----------------------

     // ---------------------- Partie servant au débogage ---------------------------------------------------------------------
-----------------------
     #ifdef DEBUG_MEM
          Compteur_Test ++;
          if (Compteur_Test >= 1000){
               Compteur_Test = 0;
               Serial.print(F("Memoire Ram dispo="));
               Serial.println(freeRam());
          }
    #endif // DEBUG
}
// ----------------------------------------------------------------------------------------------------------------------------
-----------------------
// ********************************************** FIN DU LOOP
*****************************************************************************
// ----------------------------------------------------------------------------------------------------------------------------
-----------------------

// ----------------------------------------------------------------------------------------------------------------------------
-----------------------
// ********************************************** DEBUT DES FONCTIONS
*****************************************************************************
// ----------------------------------------------------------------------------------------------------------------------------
-----------------------

// ********************** Début des fonctions système, indispensables au fonctionnement du programme
*********************************************

// ------------------------------ début fonctions associées au WEbSocket ----------------------------------------------
-----------------------
void onConnect(WebSocket &socket)
{
        #ifdef DEBUG
            Serial.println(F("onConnect called"));          // pour test
        #endif
}

// Programme de traitement des données reçues du client websocket
// --------------------- début de la fonction onData (Traitement des messages reçus ---------------------------------------
-----------------------
void onData(WebSocket &socket, char* DataString, byte FrameLength)
{
    byte Rang = 0;
    char Buff[12];

        #ifdef DEBUG
            Serial.print(F("chaine recue : "));
            Serial.write((unsigned char*)DataString, FrameLength);
            Serial.println();
        #endif

        if ((DataString[0] == (STX)) && (DataString[1] == NumCard[0]) && (DataString[2] == NumCard[1]) && (DataString[3] ==
NumCard[2]) && (DataString[4] == NumCard[3])) {
            // test si code d'accÃ¨s
            if (strncmp(&DataString[6],"PAGEC",5) == 0){
                Buff[0] = DataString[12];
                Buff[1] = DataString[13];
                Buff[2] = '\0';
                PageWeb = atoi(Buff);
                #ifdef DEBUG
                    Serial.print(F("Page : "));
                    Serial.println(PageWeb);
                #endif
            }
            if (strncmp(&DataString[6],"ACCES",5) == 0){
                #ifdef DEBUG
                    Serial.print(F("demande d'acces="));
                    Serial.println(DataString);
#endif
// test si le code d'accÃ¨s est valide
if (strncmp(&DataString[12],CODE_CARTE,4)==0){              // Réception valide du code d'accès
    Indic_Acces = 1;                                        // Autorisation d'accès
    Compteur_Deconnexion_Nb_Periode = 0;
    // Elaboration du message d'autorisation
    EvtServer[0] = STX;
    strncpy(&EvtServer[1], NumCard, 4);
    EvtServer[5] = Separator;
    strncpy(&EvtServer[6], "REACC",5);
    EvtServer[11] = Separator;
    strncpy(&EvtServer[12], "AUT",3);
    EvtServer[15] = ETX;
    EvtServer[16] = '\0';
    #ifdef DEBUG
        Serial.println(F("Code OK"));
        Serial.print(F("Chaine emise :"));
        Serial.println(EvtServer);
    #endif
}
else {
    Indic_Interdit = 1;                                     // Interdiction d'accès à la centrale
    // Elaboration du message d'interdiction
```

```
        EvtServer[0] = STX;
        strncpy(&EvtServer[1], NumCard, 4);
        EvtServer[5] = Separator;
        strncpy(&EvtServer[6], "REACC",5);
        EvtServer[11] = Separator;
        strncpy(&EvtServer[12], "INT",3);
        EvtServer[15] = ETX;
        EvtServer[16] = '\0';
        #ifdef DEBUG
            Serial.println(F("Code faux"));
            Serial.print(F("Chaine emise :"));
            Serial.println(EvtServer);
        #endif
    }
    // Vérification de la connexion du client
    if (socket.isConnected()) {
        socket.send(EvtServer,16);
    }
}
if (Indic_Acces == 0) {
    Indic_Interdit = 1;                                         // pour envoi d'une réponse d'interdiction d'accès
}
// Accès autorisé, donc applique les ordres reçus
            if (Indic_Acces==1) {
                if (strncmp(&DataString[6],"PORTA",5) == 0) {            // Réception d'une demande de commande du portail
                    commandePortail();                                  // Commande du portail
                }
                else if (strncmp(&DataString[6],"GARAG",5) == 0) {       //Réception d'une demande de commande de porte de
garage
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    Rang = atoi(Buff);
                    commandeGarage(Rang);                               // Commande de la porte de garage demandée
                }
                else if (strncmp(&DataString[6],"VOLET",5) == 0) {      // Réception d'une demande de commande des volets
                    gestion_volets(DataString);                         // appel de la fonction de la gestion des volets
                }
                else if (strncmp(&DataString[6],"DPROV",5) == 0) {      // Demande d'envoi des données de programmation
volets à l'interface web
                    envoiProgVolet();
                }
                else if (strncmp(&DataString[6],"ENPRV",5) == 0) {      // appel de la procédure de mise en mémoire des
programmations volets envoyés par l'interface web
                    enregProgrammeVolet(DataString);
                }
                else if (strncmp(&DataString[6],"DPARA",5) == 0) {      // Réception d'une demande d'envoi des paramètres
de l'unité de gestion à l'interface web
                    #ifdef DEBUG
                        Serial.println(F("Demandes des parametres"));
                    #endif
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    PageWeb = atoi(Buff);
                    if (PageWeb == 1) {                                 // Page Index
                        envoi_appairage_sondesT(5);                     // Envoi de l'appairage des sondes T° des 5
premières pièces
                        envoiObjectifElectricite();                     // Envoi de objectifs électricité pour affichage
barregraph Tendance
                    }
                    if (PageWeb == 5) {                                 // page Chauffage
                        // Demande d'envoi de l'appairage des sondes de température (par rapport au nombre de pièces de
l'habitation)
                        envoi_appairage_sondesT(12);
                    }
                    if (PageWeb == 7) {                                     // Page Alarme
                        // Demande d'envoi de la configuration et des détections de l'alarme
                        envoi_para_alarme();                                // dans module DM_protection
                    }
                    if (PageWeb == 8) {                                 // Page Electricité
                        // Demande d'envoi des compteurs et objectifs électriques
                        envoiObjectifElectricite();                     // dans module DM_electricite
                    }
                    if (PageWeb == 11) {                                // Page Paramètres
                        envoiParametresVolets();                        // Envoi des paramètres d'initialisation des
volets
                        envoi_appairage_sondesT(15);                     // Envoi de l'appairage des 15 sondes T°
                        envoiObjectifElectricite();                     // Envoi les compteurs et les objectifs
électriques
                        envoiObjectifEau();                             // Envoi les compteurs et les objectifs Eau
                        envoiEpoch();                                   // Envoi de l'heure UNIX de la centrale
                    }
                    if (PageWeb == 14) {
                        envoi_etat_cameras();
                    }
                    #ifdef CHAUFCONF
                        if (PageWeb == 18) {                                // Page radiateurs
                            envoi_mode_radia();                             // envoi du mode de fonctionnement immédiat
des radiateurs
                            envoi_program_radia();                          // envoi de la programmation des radiateurs
                        }
                    #endif
                }
                else if (strncmp(&DataString[6],"EPARA",5) == 0) {      // appel de la procédure de mise en mémoire des
paramètres envoyés par l'interface web
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    PageWeb = atoi(Buff);
                    if (PageWeb == 1) {
                        sauveParametresVolets(DataString);              // Enregistrement dans l'Eeprom de
l'initialisation des volets
```

```
                    }
                    if (PageWeb == 2) {
                        sauve_appairage_sondesT(DataString);                // Enregistrement dans l'Eeprom de l'appairage des
sondes de température
                    }
                    if (PageWeb == 3) {
                        sauveObjectifEnergie(DataString);                   // Enregistrement dans l'Eeprom des objectifs
électriques et eau (Module DM_electricite)
                    }
                }
                else if (strncmp(&DataString[6],"MDATE",5) == 0) {
                    strncpy(Buff, &DataString[12],10);
                    Buff[10] = '\0';
                    Epoch = atol(Buff);
                    modifieEpoch();
                }
                else if (strncmp(&DataString[6],"DDO48",5) == 0) {          // Demande d'envoi des données enregistrées lors
des dernières 48 heures
                    envoiDonnees48();
                }
                else if (strncmp(&DataString[6],"DINJO",5) == 0) {          // demande infos journalieres du mois specifié
                    if (Indic_Info_Jour == 0) {
                        strncpy(Annee_Recv, &DataString[12],4);
                        strncpy(Mois_Recv, &DataString[17],2);
                        envoi_info_jour();
                    }
                }
                else if (strncmp(&DataString[6],"INITI",5) == 0) {         // appel de la procédure de réinitialisation des
sondes de température
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    Rang = atoi(Buff);
                        if (Rang == 1) {                                   // 1 pour r&initialisation des sondes de température
                            initSondesT();
                        }
                }
                else if (strncmp(&DataString[6],"CMDPR",5) == 0) {          // Commande des prises électriques
                    commande_prises(DataString);
                }
                else if (strncmp(&DataString[6],"ALARM",5) == 0) {
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    Rang = atoi(Buff);
                    if (Rang == 1) {                                           // Activation ou désactivation de l'alarme
                        activeAlarme(DataString,'W');                          // W pour demande faite par l'apllication Web
                    }
                    else if (Rang == 2) {                                      // Activation ou désactivation des contacteurs
d'ouverture
                        activeContactA(DataString);
                    }
                    else if (Rang == 3) {                                      // Activation ou désactivation des détecteurs
IR
                        activeDetectA(DataString);
                    }
                }
                else if (strncmp(&DataString[6],"DJOAL",5) == 0) {            // Demande d'envoi des journaux d'évenements
de l'alarme
                    if (Indic_Journal_Alarme == 0) {
                        Buff[0] = DataString[12];
                        Buff[1] = DataString[13];
                        Buff[2] = '\0';
                        Rang = atoi(Buff);                                     // Rang définissant ie type de journal
                        envoi_journal_alarme(Rang);
                    }
                }
                else if (strncmp(&DataString[6],"EFJOA",5) == 0) {            // Demande d'effacement des journaux de
l'alarma
                    Buff[0] = DataString[12];                                  // Rang définissant ie type de journal
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    effaceJournalAlarme(Buff);
                }
                else if (strncmp(&DataString[6],"ETCAM",5) == 0) {            // Demande de modification des états de
fonctionnement des caméras
                    modif_etat_cameras(DataString);
                }
                #ifdef CHAUFCONF
                    else if (strncmp(&DataString[6],"FORAD",5) == 0) {        // Reception du mode de fonctionnement des
radiateurs
                    Buff[0] = DataString[12];
                    Buff[1] = DataString[13];
                    Buff[2] = '\0';
                    Rang = atoi(Buff);
                        if (Rang == 1) {                                       // Réception de mode de fonctionnement
immédiat des radiateurs
                            enregImmediat_Rad(DataString);
                        }
                        else if (Rang == 2) {                                  // Réception de la programmation des
radiateurs
                            enregProg_Rad(DataString);
                        }
                    }
                #endif
            }
        }
}
// ------------------------------ fin de la fonction onData ------------------------------------------------------------
----------------------

// ------------------ Début de la fonction onDisconnect (Déconnexion du webSocket) ---------------------------------------
----------------------
```

```
void onDisconnect(WebSocket &socket)
{
    #ifdef DEBUG
      Serial.println(F("onDisconnect called"));
    #endif
}
// ------------------------------ fin de la fonction onDisconnect ------------------------------------------------------------
----------------------
// ------------------------------ Fin fonctions associées au WEbSocket -------------------------------------------------------
----------------------

// ---------------  Début de la fonction ntp , retourne le nombre de sec. depuis 1 Jan 1970. (Unix time) --------------------
----------------------
unsigned long ntp()
{
    const int NTP_PACKET_SIZE= 48;                          // NTP time stamp is in the first 48 bytes of the message
    byte PacketBuffer[NTP_PACKET_SIZE];                     //buffer to hold incoming and outgoing packets

    EthernetUDP Udp;
    memset(PacketBuffer, 0, NTP_PACKET_SIZE);               //initialise toute les "case" du tableau à 0

    //construction de la requete
    PacketBuffer[0] = 0b11100011;                           // LI, Version, Mode
    PacketBuffer[1] = 0;                                    // Stratum, or type of clock
    PacketBuffer[2] = 6;                                    // Polling Interval
    PacketBuffer[3] = 0xEC;                                 // Peer Clock Precision
    // 8 bytes of zero for Root Delay & Root Dispersion
    PacketBuffer[12]  = 49;
    PacketBuffer[13]  = 0x4E;
    PacketBuffer[14]  = 49;
    PacketBuffer[15]  = 52;

    Udp.begin(PORT_TIME_SERVER);
    Udp.beginPacket(TIME_SERVER, 123);                      // NTP requests are to port 123
    Udp.write(PacketBuffer,NTP_PACKET_SIZE);
    Udp.endPacket();

    delay(1000);
        if ( Udp.parsePacket() )
        {
            Udp.read(PacketBuffer,NTP_PACKET_SIZE);
            Udp.stop();
            const unsigned long seventyYears = 2208988800UL;
            unsigned long highWord = word(PacketBuffer[40], PacketBuffer[41]);
            unsigned long lowWord = word(PacketBuffer[42], PacketBuffer[43]);
            unsigned long secsSince1900 = highWord << 16 | lowWord;
            return secsSince1900 - seventyYears;
        }
        else
        {
            lcd.clear();                                    // efface LCD
            lcd.setCursor(0,0);                             // positionne curseur colonne 1, ligne 1
            lcd.print("Pb NTP");
        }
    return 0;
}
// --------------------------- fin de la fonction ntp  ---------------------------------------------------------------------
----------------------

// --------------- début de la fonction fonction recherche NTP et mise à jour RTC -----------------------------------------
----------------------
void calcul_ntp_time(unsigned long Epoch)
{
    lcd.setCursor(8,3);                                 // positionne curseur colonne 10, ligne 4
        if (Epoch == 0)
        {
            lcd.print("Defaut NTP  ");
        }
        else
        {
            calcul_DLS();
            // Remets à l'heure le RTC DS1307
            RTC.set(Epoch + 3600 + (Dls * 3600));
            Compteur_Ntp = 0;                             // remise à jour du compteur NTP
        }
}
// --------------- fin de la fonction fonction recherche NTP et mise Ã  jour RTC -------------------------------------------
----------------------

// --------------- début de la fonction calcul DLS -------------------------------------------------------------------------
----------------------
// fonction qui calcule DLS (Day Light Saving : heure d'hiver DLS = 0 , d'heure d'été DLS = 1)void calculDLS()
void calcul_DLS()
{
    if ((Mois > 3) && (Mois < 10) || ((Mois == 3) && (DateJour > 24) && (((DateJour - JourSemaine) > 24) || (JourSemaine ==
7)))
         || ((Mois == 10) && ((DateJour < 25) || (((DateJour - JourSemaine) < 25) && (JourSemaine != 7)))))
    {
        Dls = 1;
    }
    else
    {
        Dls = 0;
    }
}
// --------------- Fin de la fonction calcul DLS ---------------------------------------------------------------------------
----------------------

// --------------- début de la fonction lecture_rtc ------------------------------------------------------------------------
----------------------
// Fonction de lecture du module Horloge DS1307RTC avec mise à jour des variables Temps
void lectureRTC()
{
```

```
    Epoch_Local = RTC.get();
    Hours = hour(Epoch_Local);
    Mins = minute(Epoch_Local);
    Sec = second(Epoch_Local);
    DateJour = day(Epoch_Local);
    Mois = month(Epoch_Local);
    Annee = year(Epoch_Local);
    JourSemaine = weekday(Epoch_Local);
}
// -------------------------- fin de la fonction lecture RTC ------------------------------------------------------------
----------------------

// --------------------début de la fonction affiDateHeure (affichage de la date et de l'heure dans le LCD) -----------------
----------------------
void affiDateHeure(int Ligne)
{
    int AnneeCours;

      lcd.setCursor(0,Ligne);                // positionne le curseur colonne 1, ligne demandée
    // affichage jour de la semaine
    switch (JourSemaine)
    {
        case 1:
            lcd.print("Dim. ");
            break;
        case 2:
            lcd.print("Lun. ");
            break;
        case 3:
            lcd.print("Mar. ");
            break;
        case 4:
            lcd.print("Mer. ");
            break;
        case 5:
            lcd.print("Jeu. ");
            break;
        case 6:
            lcd.print("Ven. ");
            break;
        case 7:
            lcd.print("Sam. ");
            break;
    }
    // affichage jour
        if (DateJour < 10)
        {
            lcd.print('0');
        }
    lcd.print(DateJour);
    lcd.print('/');
    // affichage mois
        if (Mois < 10)
        {
            lcd.print('0');
        }
    lcd.print(Mois);
    lcd.print('/');
      // affichage année
      AnneeCours = Annee - 2000;
      lcd.print(AnneeCours);
      //
    lcd.print(' ');
    // affichage heures
        if (Hours < 10)
        {
            lcd.print('0');
        }
    lcd.print(Hours);
      lcd.print(':');
      // affichage minutes
        if (Mins <10)
        {
            lcd.print('0');
        }
    lcd.print(Mins);
}
// ------------------------------- fin de la fonction affiDateHeure ----------------------------------------------------
----------------------

// -------------------------- début de la fonction affichage du mode de fonctionnement LCD ligne 2 -------------------------
----------------------
static inline void affichageMode()
{
    lcd.setCursor(0, 1);                        // positionne curseur colonne 1, ligne 2
        if (TempsLED == TEMPS_LED_NORMAL) {
            lcd.print("Mode Normal   ");
        }
        else {
            lcd.print("Mode Defaut   ");
        }
    // affichage de la température du boitier
    lcd.print(TempT[15]/10);
}
// ----------- ----------- fin de la fonction affichage du mode de fonctionnement LCD ligne 2 ------------------------------
----------------------

// ---------------------- début de la fonction testLDR (test de la luminosité) -------------------------------------------
----------------------
static inline void testLDR()
{
        if (analogRead(PIN_LDR) > 500) {        // mesure de l'éclairement de la LDR
            if (TempsMn > TempsLDR) {           // tempo max 1 minute
```

```
                digitalWrite (PIN_BLLCD,0);     // Back Light LCD éteint
            }
        }
        else{
            digitalWrite (PIN_BLLCD,1);         // Back Light LCD allumé
            TempsLDR = TempsMn ;
        }
}
// ------------------------ fin de la fonction testLDR ------------------------------------------------------------------------
----------------------

// --------------------- début de la fonction blinkLED (fait clignoter la LED) ---------------------------------------------
----------------------

void blinkLED(int TempsClign)
{
    static unsigned long PrevMillis = 0;        // déclaration d'une variable pour la temporisation de la LED
    static boolean EtatLed = true;              // déclaration d'une variable pour la temporisation de la LED

    unsigned long CurMillis = millis();
        if(CurMillis - PrevMillis > TempsClign) {
            PrevMillis = CurMillis;
                if(EtatLed) {
                    digitalWrite (PIN_LED_WATCHDOG,LOW);
                    EtatLed = false;
                }
                else {
                    digitalWrite (PIN_LED_WATCHDOG,HIGH);
                    EtatLed = true;
                }
        }
}
// ------------------------ fin de la fonction blinkLED ------------------------------------------------------------------------
----------------------

// ---------------------------- début de la fonction test carte SD ----------------------------------------------------------
----------------------
void testCarteSD()
{
    lcd.setCursor(0,3);                     // positionne curseur colonne 1, ligne 4
    TestSD = SD.begin(PIN_SELECT_SD);       // initialisation de la carte SD avec broche 4 en tant que CS - renvoie true/false
        if (TestSD != true) {               // si initialisation n'est pas réussie
            lcd.print("SD Pb");
            CarteSD = false;
            TempsLED = TEMPS_LED_PANNE;     // fonctionnement anomalie
        }
        else {                              // si initialisation rÃ©ussie
            lcd.print("SD OK");
            CarteSD = true;
        } // fin si SD.begin
}
// ---------------------------------- fin de la fonction test catre SD ------------------------------------------------------
----------------------

// ------------------------------ début de la fonction Reset Watchdog ---------------------------------------------------------
----------------------
void watchdog()
{
    digitalWrite (PIN_WATCHDOG, HIGH);
    delay (2);
    digitalWrite (PIN_WATCHDOG, LOW);           // Reset Watchdog
}
// ---------------------------------------- fin de la fonction Reset Watchdog ------------------------------------------------
----------------------

// --------------- Début de la fonction envoiEpoch (envoi de la date et de l'heure au serveur) ------------------------------
----------------------
void envoiEpoch()
{
    calcul_DLS();
    Epoch = Epoch_Local - 3600 - (3600 * Dls);

    EvtServer[0] = STX;
    strncpy(&EvtServer[1], NumCard, 4);
    EvtServer[5] = Separator;
    strncpy(&EvtServer[6], "DATHE",5);
                EvtServer[11] = Separator;
                snprintf(&EvtServer[12],11,"%010lu",Epoch);             //Heure UTC/GMT.
                EvtServer[23] = ETX;
                EvtServer[24] = '\0';
                //Vérification de la connexion du client.
                if (wsServer.isConnected()) {
                      wsServer.send(EvtServer,24);                      //Envoi date-heure
                }
}
// --------------- Fin de la fonction envoiEpoch (envoi de la date et de l'heure au serveur) --------------------------------
----------------------

// ----------------- Début de la fonction modifieEpoch (modifie l'heure Unix envoyée par le serveur) -------------------------
----------------------
void modifieEpoch()
{
    calcul_DLS();
    // Remets à l'heure le RTC DS1307
    RTC.set(Epoch + 3600 + (Dls * 3600));

    creMessage("04","Heure OK",7);
}
// ----------------- Fin de la fonction modifieEpoch (modifie l'heure Unix envoyée par le serveur) ---------------------------
----------------------

// ---------------- Début de la fonction sauverInt (écriture d'un type int en mémoire EEPROM) -------------------------------
----------------------
```

```
void sauverInt(int adresse, int val)
{
    //découpage de la variable val qui contient la valeur à sauvegarder en mémoire
    unsigned char faible = val & 0x00FF;            //récupère les 8 bits de droite (poids faible) -> 0010 1100
    //calcul : 1101 0111 0010 1100 & 0000 0000 1111 1111 = 0010 1100

    unsigned char fort = (val >> 8) & 0x00FF;       //décale puis récupère les 8 bits de gauche (poids fort) -> 1101 0111
    //calcul : 1101 0111 0010 1100 >> 8 = 0000 0000 1101 0111 puis le même & qu'avant

    //puis on enregistre les deux variables obtenues en mémoire
    EEPROM.write(adresse, fort) ;                   //on écrit les bits de poids fort en premier
    EEPROM.write(adresse+1, faible) ;               //puis on écrit les bits de poids faible à la case suivante
}
// ---------------- Fin de la fonction sauverInt (écriture d'un type int en mémoire EEPROM) ---------------------------------
------------------------

// ------------ Début de la fonction lireInt (lecture de la variable de type int enregistrée précédemment par la fonction
sauveInt) -----------------
int lireInt(int adresse)
{
    int val = 0 ;                                   //variable de type int, vide, qui va contenir le résultat de la lecture

    unsigned char fort = EEPROM.read(adresse);      //récupère les 8 bits de gauche (poids fort) -> 1101 0111
    unsigned char faible = EEPROM.read(adresse+1);  //récupère les 8 bits de droite (poids faible) -> 0010 1100

    //assemblage des deux variable précédentes
    val = fort ;                                    // val vaut alors 0000 0000 1101 0111
    val = val << 8 ;                                // val vaut maintenant 1101 0111 0000 0000 (décalage)
    val = val | faible ;                            // utilisation du masque
    // calcul : 1101 0111 0000 0000 | 0010 1100 = 1101 0111 0010 1100

    return val ;                                    //on n'oublie pas de retourner la valeur lue !
}
// ------------ Fin de la fonction lireInt (lecture de la variable de type int enregistrée précédemment par la fonction
sauveInt) -------------------

// -------------------------- Début de la fonction freeRam : Calcul mÃ©moire ram dynamique ---------------------------------
------------------------
int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
// ------------------- fin de la fonction freeRam -----------------------------------------------------------------------
------------------------
// -------------------------------------------------------------------------------------------------------------------
------------------------
// ********************************** FIN DES FONCTIONS DU PROGRAMME SYSTEME
*************************************************************
// -------------------------------------------------------------------------------------------------------------------
------------------------
// ------------------------------------------------------------------------------------------------------------------------
------------------------
// ***************************** DEBUT DES FONCTIONS COMMUNES A PLUSIEURS MODULES
*************************************************************
// -------------------------------------------------------------------------------------------------------------------
------------------------
void creMessage(char* NumRang, char* Message, byte LongMes)
{
    EvtServer[0] = STX;
    strncpy(&EvtServer[1], NumCard, 4);
    EvtServer[5] = Separator;
    strncpy(&EvtServer[6], "MESSA",5);
    EvtServer[11] = Separator;
    strncpy(&EvtServer[12], NumRang, 2);
    EvtServer[14] = Separator;
    strncpy(&EvtServer[15], Message, LongMes);
    EvtServer[15 + LongMes] = ETX;
    EvtServer[16 + LongMes] = '\0';
    wsServer.send(EvtServer,16 + LongMes);
        #ifdef DEBUG
            Serial.print(F("chaine emise = "));
            Serial.println(EvtServer);
        #endif
}
// ------------------------------- fin de la fonction creMessage ---------------------------------------------------------
------------------------

// ---------------- Début de la fonction read_reception_relink (lecture de la réception des données du module RFLink ---------
------------------------
void read_reception_rflink()
{
    char DebSignal = 0x0a;                                  // caractère LF en hexadécimal
    char FinSignal = 0x0d;                                  // caractère CR en héxadécimal
    byte i;

    if (Serial2.available() > 0) {
        InByte2 = Serial2.read();
            if (InByte2 == DebSignal) {
                Bufflen2 = 0;
            }
            else {
                Buffin2[Bufflen2] = InByte2;
                Bufflen2++;
            }
        if (Bufflen2 > 80) {
            Bufflen2 = 0;                       // traitement erreur
        }
        else if(InByte2 == FinSignal && Bufflen2 > 5) {     // fin de ligne trouvée
            Buffin2[Bufflen2] = '\0';                       // caratère de fin de chaine
                #ifdef DEBUG
```

```
                    Serial.print(F("RFLink = "));
                    Serial.write(Buffin2);
                    Serial.println();
                #endif
            traiteRFLink(Buffin2);
        }
    }
}
// ---------------- Fin de la fonction read_reception_relink (lecture de la réception des données du module RFLink -----------
----------------------

// ------------------ Début de la fonction traiteRFLink (traitement des données reçues du module RFLink) -------------------
----------------------
void traiteRFLink(char* ChRecue)
{
    // Interprétation de la chaine reçue
        if (strncmp(&ChRecue[0],"20;",3) == 0){                          // Réception d'une donnée valide RFLink
            if ((strncmp(&ChRecue[6],"Oregon Temp;",12) == 0) || (strncmp(&ChRecue[6],"Oregon TempHygro",16) == 0)) {
// Réception d'une donnée valide RFLink
                traiteSondesTemperature(ChRecue);                     // dans module DM_temperature
            }
            else if ((strncmp(&ChRecue[6],"Chuango",7) == 0) || (strncmp(&ChRecue[6],"Xcellent",8) == 0)){             //
marque des différents capteurs alarme
                traiteAlarme(ChRecue);                                // dans module DM_protection
            }
        }
        else if (strncmp(&ChRecue[0],"50;",3) == 0) {                  // Réception d'une donnée du clavier déporté
            if (strncmp(&ChRecue[3],"NRFcl;",6) == 0) {                // Identification du clavier
                if (strncmp(&ChRecue[9],"ETAT",4) == 0) {              // Demande d'envoi de l'état de l'alarme
                    envoi_etatAlarme();
                }
                else if (strncmp(&ChRecue[9],"METAT",5) == 0) {
                    activeAlarme(ChRecue, 'C');                                  // C pour demande faite par le clavier
déporté
                }
            }
        }
}
// ------------------ Fin de la fonction traiteRFLink (traitement des données reçues du module RFLink) ----------------------
----------------------

// ------------------------------------------------------------------------------------------------------------------------
----------------------
// ***************************** FIN DES FONCTIONS COMMUNES A PLUSIEURS MODULES
*****************************************************************
// ------------------------------------------------------------------------------------------------------------------------
----------------------
```