REL4YLC - Code Arduino

```
/*************************************************************************************
REL4YLC  V3.3
auth: Yves Le Chevalier  -  Maj le 02/01/2017
Prog de suivi météo (température, taux d'humidité, pression barométrique) et télécommande de relais par internet
Config : carte Arduino Mega 2560 + shield ethernet Arduino + shield 4 relais Microbot + Ecran LCD série I2C (2x16 car)
     + module RTC/Eeprom I2C + mod. BMP085 I2C (pression) + conv.anal. I2C 3,3v.<->5v.
     + mod.DHT22 (hygro+temp) + inter coupe reset + 3 dip-swithes de commande affichages LCD
     + émetteur RF433 Mhz (prise 220v radiocommandée) + récepteur RF433 Mhz (réception temp.extérieure)
     Module extérieur : Atmega328P-PU + MR003(TCN75A) en I2C + émetteur RF433 Mhz
-------------------------------------------------------------------------------------
                                Affectation des entrées-sorties :
0  serial (RX)
1  serial (TX)
4  réservé SD card du shield ethernet
5  relais 1 du shield relais (out)
6  relais 2 du shield relais (out)
7  relais 3 du shield relais (out)
8  relais 4 du shield relais (out)
9  Data à envoyer sur émetteur radio (pour prises télécommandées)
10  réservé au W5100 du shield ethernet
18  (ie5) non utilisé
19  (ie4) interruption n° 4 pour le récepteur radio
20  (ie3) I2C - SDA
21  (ie2) I2C - SCL
23  LED rouge témoin réception radio de temp.ext. (out)
24  LED jaune témoin connexion client ethernet (out)
30  Déclencheur du reset de la carte (out)  Connecté à la pin reset avec un inter en série (pour load prog)
32  Allumage LCD sur écran 1
34  Allumage LCD sur écran 2
36  Allumage LCD sur écran 3
40  Data du DHT22 (in)
50  Bus SPI (Ethernet shield) MISO
51  Bus SPI (Ethernet shield) MOSI
52  Bus SPI (Ethernet shield) SCK
53  Bus SPI (Ethernet shield) SS (à mettre en OUTput)
-------------------------------------------------------------------------------------
                                Historique des évolutions
v1.2 : Commandes des relais (swith on/off) et accès aux fichiers protégés par mot de passe à durée limitée (1 minute).
v1.3 : Enregistrement historique des commandes des relais dans fichier HIACTIV.txt sur carte SD.
v1.4 : Récup heure sur serveur NTP (ntp.internet-fr.net) une fois par 24h pour mise à jour de l'horloge DS1307.
v1.5 : Afficher date/heure,humidité,température intérieure,température extérieure,pression atmosphérique.
```

v1.6 : Enregistrer pression et temp toutes les 3 heures dans fich. MESURES.txt + affichage variations pression depuis 48h.
v1.7 : Commande d'affichage historique des pressions et temperatures entregistrées depuis le début.
v1.8 : Mettre fichier HIACTIV.txt au format CSV + intégrer hologe temps réel ds1307 + ajouter un écran LCD
v1.9 : Mémoriser timbre enregistrement pression/temp (MESURES.txt) dans Eeprom du DS1307 (1 enregt / 3h)
+ bouton interrupteur d'affichage des mesures sur LCD + temporiser mesures DHT11 une fois par minute
v2.0 : Stocker le mot de passe de référence (6 car) dans fichier PARAM.txt sur carte SD (à modidier sur ordi).
v2.1 : Commande affichage historique des commandes + Stocker timbre dernière synchro serveur NTP dans Eeprom du DS1307
+ commande d'effacement du fichier HIACTIV.txt (demande d'un second mot de passe).
v2.2 : LM35 pour temp intérieure et TNC75 (i2c) pour temp extérieure. Temporiser mesures toutes les 60 secondes.
v2.3 : Mise en mémoire flash des messages serveur (F) + Reset apres connex. serveur NTP (24h) ou si manque de mémoire Sram
+ Ajout d'un interrupteur entre pin 30 et pin Reset pour permettre de téléverser le programme.
v2.4 : Affichage tournant sur LCD (appuis successifs sur bouton) pour contrôle des paramètres de fonctionnement
+ allumer led de visualisation quand un client est connecté.
v2.5 : Vérifier toutes les 6 heures la variation barométrique et envoyer un tweet si alerte météo (baisse > 1 Hpa par heure)
+ Enregitrer trace des resets système dans le fichier HIACTIV.txt.
v2.6 : Après redémarrage restaurer les relais dans leur état initial avant le reset.
v2.7 : Découpler le calcul du timbre d'horodatage de la synchro horloge avec le serveur NTP
+ lisser relevé de temp intérieure sur moyenne de 10 mesures
v2.8 : Envoi commande radio on/off (pour prises radiocommandées) synchrones avec commande relais n°1, 2 et 3
et réception radio de la température extérieure (CF : Sonde_temp_radio.ino) + led témoin réception radio
v2.9 : Gestion temp.ext. négatives + enregt fichier mesures en format fixe (35 car.) + visu état des relais en-tête écran
v3.0 : Affichage historique mesures du mois en cours ou tout l'historique + affichage en tableaux prop. à largeur fenêtre.
v3.1 : Archivage sur demande des mesures des mois précédents dans un fichier archives (mot de passe niveau 2)
+ codage data radio pour différencier commandes relais et transmission des températures.
v3.2 : remplacer DHT11 par DHT22 et mesurer température intérieure et humidité avec le DHT22 => Supprimer le LM35.
v3.3 : allumer diode rouge en continu si pas de réception radio de la sonde extérieure depuis plus de 10 min
+ remplacer bouton commande LCD par 3 switches. (suppression de l'interruption sur pin 18)

---

*/

```
#define TRACES 0          // 1 pour activer trace sur port série, 0 pour la désactiver
//
```

```
#include <stdlib.h>
```

```
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <Twitter.h>
#include <RTClib.h>        // pour utilisation horloge DS1307
#include <avr/wdt.h>       // lib pour forcer le reset
#include <Time.h>
#include <SD.h>
#include <Wire.h>
#include <RCSwitch.h>
RCSwitch radio = RCSwitch();    // objet radio = émetteur et récepteur radio fréquence 433 Mhz
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x3F,2,1,0,4,5,6,7); // déclaration afficheur LCD série I2C
#include <Adafruit_BMP085.h>   // capteur de pression atmosphérique (température non exploitée)
Adafruit_BMP085 bmp085;        // branché en I2C sur pin 20 (SDA) et 21 (SCL) - sur Méga
#include <dht.h>            // capteur humidité et température de la pièce
RTC_DS1307 RTC;
dht DHT;
static byte mac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };  // mac-adresse de la carte ethernet
IPAddress ip(192,168,0,70);   //  IP locale (adresse DHCP forcée sur freebox en fontion adresse
mac)
IPAddress serveurntp(212,37,192,31);  // adresse IP serveur NTP : ntp.internet-fr.net
Twitter twitter("36022846-qWGkQZu1U4lOq1RtsLkZaZqsQ6X3BMG5c8STDXl5L");   // token
pour poster sur mon compte twitter (privé)
EthernetServer server(80);       //server port 80
EthernetUDP Udp;
unsigned int portudp = 8888;    // port UDP local sur le routeur (rerouté sur port externe 123 pour
serveur NTP)
#define DHT22PIN 40   // data du DHT22 en entrée sur pin digitale 40
#define out1 5        // relais 1 (pin 5 arduino)
#define out2 6        // relais 2 (pin 6 arduino)
#define out3 7        // relais 3 (pin 7 arduino)
#define out4 8        // relais 4 (pin 8 arduino)
#define ledrx 23      // led temoin réception radio (temp ext)
#define ledconn 24    // led temoin connexion client ethernet
#define swlcd1 32     // switch LCD affichage écran 1
#define swlcd2 34     // switch LCD affichage écran 2
#define swlcd3 36     // switch LCD affichage écran 3
String readString = "";
boolean pwdok1,pwdok2,deconn,command,indhimmc,indhimto,indhisco,actio2,indeff,indarch =
false;
unsigned int rel1,rel2,rel3,rel4,ireset = 0;   // indic état des relais 0/1    (et reset système pour
stockage activité)
unsigned int eprel1,eprel2,eprel3,eprel4 = 0;    // indic dernier état relais enregistré
unsigned long heurdeb,duree,syncntp = 0;
unsigned long duree1900 = 0;       // Nb de secondes depuis 01/01/1900 à 00:00:00
File fichier;   // objet fichier
File fichier1;  // objet fichier 1
File fichier2;  // objet fichier 2
char tempchar[5];
char hygchar[2];
```

```
float hygro;
byte enreg[35];    // buffer enregistrement  (reste un filler de 10 car)
int codret;      // code retour de fonction
int sdko,sdlef1,sdecf1,sdecf2,sdlef2,sdlef3 = 0;   // indicateurs erreur lect/ecrit sur carte SD
const int taillentp= 48;   // timestamp dans les 48 premiers caracteres reçus du serveur NTP
byte bufferntp[taillentp];  // buffer de stockage des paquets reçus et émis sur port UDP
String timbre(14);        //  timbre pour horodatage des enregistrements
int decalhor = 2;        // decalage horaire pour l'heure locale en été (=1 en hiver)
time_t tempsposix;       // stockage du temps posix récupéré sur serveur NTP
int humip;        // humidite pièce
float tempint,tempext = 0.0;      // températures interieure et exterieure
unsigned long secondactu,secondenba,secondcapt,secondvapr,secondtempex=0;  // stockage
secondes (pour calcul interval)
byte eeptimbre [4] = {0};     // pour stockage timbre en eeprom du DS1307
int numenr[6]={2,3,5,9,13,17};   // n° des enregt à lire dans fichier MESURES
String valpress[6];      // valeurs des pressions lues dans fichier MESURES
int vario[6];          // valeurs des variations de pression par rapport à l'actuelle
int presact,prespre;   // pour calcul variations pressions
int ann1,moi1,jou1,heu1,min1,sec1;  // pour maj de l'horloge RTC
boolean rtcko,majrtc = false;    // indicateur de dysfonctionnement et de MaJ de l'horloge
int afflcd,afflcdpre = 0;    // indicateurs de mise en fonction de l'afficheur LCD
unsigned long timboupre,timbouact=0;  // pour éviter le rebond sur appui bouton
unsigned long milprelcd=0;       //  pour tempo affichage LCD
String motpas1,motpas2,passwd = "";    // mot de passe pour contrôle accès aux commandes
int nivosecu = 1;     // niveau de sécurité pour controle mot de passe (1er niv. par défaut)
int couleur;       // couleur du message d'information météo
int nbconn = 0;    // nbre de connexions clients durant la session
int tweetok = 0;   // indicateur d'envoi de tweet alerte météo
//_____________________________________________________________________________
String versionprog = "v3.3";            // Version du programme
//_____________________________________________________________________________
void setup() {
  digitalWrite(30, HIGH);  // à faire dès le démarrage du prog pour éviter de boucler sur le reset
  pinMode(30, OUTPUT);     // broche 30 en sortie  pour commander le reset de la carte
  #if TRACES
    Serial.begin(9600);
    Serial.print("REL4YLC -  ");
    Serial.println(versionprog);
    Serial.print("Memoire SRAM au setup : ");
    Serial.println(freeRam());
  #endif
  pinMode(swlcd1, INPUT);    // swithes commande LCD
  pinMode(swlcd2, INPUT);
  pinMode(swlcd3, INPUT);
  pinMode(53, OUTPUT);     //  broche SS en sortie pour SPI du shield ethernet
  pinMode(ledrx, OUTPUT);      // led témoin de réception radio (sonde ext.)
  pinMode(ledconn, OUTPUT);      // led témoin de connexion client
  digitalWrite(ledrx, HIGH);    // coupe la led témoin radio
  digitalWrite(ledconn, HIGH);   // coupe la led témoin client
  pinMode(out1, OUTPUT);      // mettre les relais en sortie
  pinMode(out2, OUTPUT);
  pinMode(out3, OUTPUT);
```

```
  pinMode(out4, OUTPUT);
  digitalWrite(out1, LOW);   // tous les relais coupés au depart
  digitalWrite(out2, LOW);
  digitalWrite(out3, LOW);
  digitalWrite(out4, LOW);
  if(Ethernet.begin(mac) == 0) {    // démarrage ethernet par DHCP
   Serial.println("Erreur de connexion Ethernet par DHCP");
   while(true) ;  // Boucle sans fin puisque prog ne peut pas fonctionner sans ethernet
  }
  delay(500);    // Laisse le temps à la carte ethernet pour s'initialiser
  IPAddress myIPAddress = Ethernet.localIP();
  #if TRACES
    Serial.print("Addresse IP : ");  // toujours (192,168,0,70) (cf routeur Freebox BAUD perm)
    Serial.println(myIPAddress);
  #endif
  server.begin();
  Udp.begin(portudp);
  bmp085.begin();
  Wire.begin();
  RTC.begin();
  if (! RTC.isrunning()) {
    rtcko = true;
    #if TRACES
      Serial.println("Horloge RTC pas en fonction");
    #endif
  }
  #if TRACES   //  date et heure du ds1307 pour contrôle
    affheurserie();
  #endif
  lcd.begin(16,2);    // definition de l'afficheur 2 lignes x 16 car.
  lcd.setBacklightPin(3,NEGATIVE);  // déclaration du rétroeclairage afficheur
  lcd.setBacklight(HIGH);     // éteindre le rétroeclairage au départ
  initsdcard();
  delay(300);      // ne pas aller trop vite, c'est un arduino !   :-))
  lecpasswd();      // récupérer mots de passe niveau 1 et 2 sur carte SD
  for (int i=0; i<4; i++) {  // récupérer timbre du dernier enregt pression et temp. (pos 0 à 3 eeprom)
    eeptimbre[i] = RTC.readnvram(i);
  }
  secondenba = ( ((unsigned long)eeptimbre[0] << 24)      // conversion des 4 bytes en un long
              + ((unsigned long)eeptimbre[1] << 16)
              + ((unsigned long)eeptimbre[2] << 8)
              + ((unsigned long)eeptimbre[3] ) );
  #if TRACES
    Serial.print("Lecture timbre dern.stockage Barom : ");
    Serial.println(secondenba);
  #endif
  for (int i=4; i<8; i++) {   // recupérer timbre de derniere synchro avec serveur NTP (pos 4 à 7
eeprom)
    int j = i-4;
    eeptimbre[j] = RTC.readnvram(i);
  }
  syncntp = ( ((unsigned long)eeptimbre[0] << 24)      // conversion des 4 bytes en un long
```

```
          + ((unsigned long)eeptimbre[1] << 16)
          + ((unsigned long)eeptimbre[2] << 8)
          + ((unsigned long)eeptimbre[3] ) );
  #if TRACES
    Serial.print("Lecture timbre dern. synchro serveur NTP : ");
    Serial.println(syncntp);
  #endif
  for (int i=8; i<12; i++) {    // recupérer timbre de dernier test de variation de pression (pos 8 à 11
eeprom)
    int j = i-8;
    eeptimbre[j] = RTC.readnvram(i);
  }
  secondvapr = ( ((unsigned long)eeptimbre[0] << 24)      // conversion des 4 bytes en un long
            + ((unsigned long)eeptimbre[1] << 16)
            + ((unsigned long)eeptimbre[2] << 8)
            + ((unsigned long)eeptimbre[3] ) );
  #if TRACES
    Serial.print("Lecture timbre dern. test varia pression : ");
    Serial.println(secondvapr);
  #endif
  lecderact();      // lecture dernier état des relais
  rel1 = eprel1;
  rel2 = eprel2;
  rel3 = eprel3;
  rel4 = eprel4;
  #if TRACES
    Serial.print("restauration etat relais : ");
    Serial.print(rel1);
    Serial.print(rel2);
    Serial.print(rel3);
    Serial.println(rel4);
  #endif
  activrel();      // restaurer les relais à leur état avant reset
  ireset=9;       // enregistrer indic de démarrage ou de reset de la session
  enrtraceact();    // enregistrer trace du démarrage dans fichier HIACTIV
  ireset=0;
  radio.enableTransmit(9);    // data emetteur radio sur pin PWM 9
  radio.setRepeatTransmit(5);  // nombre de répétitions de l'envoi du code
  radio.enableReceive(4);    // interruption n°4 récepteur radio sur la pin 19 sur Mega
}
//_____________________________________________________________________________
void loop() {
  if ((majrtc == true) || (freeRam()<2048)) {  // faire un reset si < 2k de mém.Sram ou si accès récent
au serveur NTP
    #if TRACES
      Serial.println("_____________________________________________");
      affheurserie();
      delay(100);
      Serial.println(majrtc);
      Serial.print("Reboot du systeme     Sram restante = ");
      Serial.println(freeRam());
    #endif
```

```
    digitalWrite(30, LOW);    // active le RESET de la carte (en le mettant au niveau bas)
   }
   receptradio();            // réception temp extérieure
   delay(100);      // ne pas aller trop vite, c'est un arduino !   :-))
   DateTime now = RTC.now();
   secondactu = now.unixtime();
   if ((unsigned long)(secondactu - secondtempex) > 600) { // test si plus de 10 minutes depuis
dernier réception temp. extérieure
    digitalWrite(ledrx, LOW);    // allumer diode rouge pour alerte
   }
   if ((unsigned long)(secondactu - secondcapt) >= 60) {   // relevé des capteurs toutes les 60 sec.
    #if TRACES
     Serial.println("mesure humidite + temp. DHT22");
    #endif
    mesuredht22();                  // mesure humidité et température intérieure
    delay(100);      // ne pas aller trop vite, c'est un arduino !   :-))
    #if TRACES
     Serial.println("mesure pression BMP085");
    #endif
    delay(100);      // ne pas aller trop vite, c'est un arduino !   :-))
    presact = (bmp085.readPressure()+700)/100;   // pression barometrique actuelle
    secondcapt = secondactu;
    #if TRACES
     Serial.println("Releve des capteurs");
     Serial.print("Humiditee : ");
     Serial.print(humip);
     Serial.print("     Temp.int. : ");
     Serial.print(tempint);
     Serial.print("     Temp.ext. : ");
     Serial.print(tempext);
     Serial.print("     Pression. : ");
     Serial.println(presact);
     Serial.print("Memoire SRAM libre : ");
     Serial.println(freeRam());
    #endif
   }
   affichlcd();      // vérifier si demande affichage infos sur LCD
   if ((unsigned long)(secondactu - secondenba) >= 10800) { // test si 3 heures écoulées depuis
dernier enreg pression
    stockmesures();       // enregistrer timbre, pression, temp et hygro. dans fichier MESURES
    if (sdecf2 == 2) {      // si enrgt OK
     secondenba = secondactu;
     memoeeprom(0);            // stocker timbre enregt dans Eeprom du DS1307 (pos 0 à 3)
     delay(500);      // ne pas aller trop vite, c'est un arduino !   :-))
    }
   }
   if ((unsigned long)(secondactu - secondvapr) >= 21600) { // test si 6 heures écoulées depuis
dernier test de variation
    ctrlvarbaro();   // contrôle de la variation de pression
    secondvapr = secondactu;
    memoeeprom(8);            // stocker timbre du test dans Eeprom du DS1307 (pos 8 à 11)
    delay(500);      // ne pas aller trop vite, c'est un arduino !   :-))
```

```
  }
  if ((unsigned long)(secondactu - syncntp) > 86400) { // on synchronise l'horloge avec serveur NTP
une fois par jour.
   majhorloge();
  }
  EthernetClient client = server.available();
  if (client) {                // test si connexion internet
    digitalWrite(ledconn, LOW);   // allumer led témoin connexion
    nbconn += 1;          // comptage des connexions
    controlduree();     // verification si password expiré
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if (readString.length() < 100)  readString += c;  // concatenation des caracteres lus
        if (c == '\n') {           // fin de message
          if (nivosecu == 1) {
            ctrlpwd1(client);         // test si password de premier niveau saisi
            if (pwdok1) {
              action1();          // executer les commandes
              afficomm(client);
              if (indhimmc || indhimto)   affhistmesur(client);    // afficher historique des mesures
              if (indhisco)   affhistcomm(client);          // afficher historique des commandes
              affich2(client);        // affichage de la page des commandes
            }
            else {
              afficomm(client);
              affich1(client);     // affichage demande du mot de passe
            }
          }
          if (nivosecu == 2) {
            ctrlpwd2(client);         // test si password de second niveau saisi
            if (pwdok2) {
              action2(client);          // exécuter les commandes de niveau 2
              afficomm(client);
              affich2(client);     // ré-affichage de la page des commandes
            }
            else {
              afficomm(client);
              affich1(client);    // affichage demande du mot de passe
            }
          }
        }
      }
    }
  }
  else digitalWrite(ledconn, HIGH);   // éteindre led témoin connexion
}
//_____________________________________________________________________________
void receptradio() {
  if (radio.available()) {       // reception radio
    float codrad = radio.getReceivedValue();
    digitalWrite(ledrx, LOW);   // allumer led témoin réception radio
```

```
    delay(20);
    digitalWrite(ledrx, HIGH);
    radio.resetAvailable();
    if (codrad >= 5555555 && codrad <= 5568555) {    // code temp. valide entre 5555555 (0°)  et 5568555 (-30°)
      tempext = codrad - 5555555;   // translation de la valeur pour ne pas interférer avec les codes des relais
      if (tempext > 10000)  tempext = ((tempext - 10000) * -1) /100;   // conversion temp négative (codée +10000)
      else tempext = tempext / 100;        // conversion temp positive
      DateTime now = RTC.now();
      secondtempex = now.unixtime();
      #if TRACES
        Serial.print("Reception radio : ");
        Serial.print(codrad);
        Serial.print(" /  temp. = ");
        Serial.println(tempext);
      #endif
    }
  }
}
//_____________________________________________________________________________
//void interlcd()  {           // interruption par appui sur bouton
//  timbouact = millis();
//  if (timbouact - timboupre > 500) {    // temporisation appui sur bouton
//    afflcd += 1;           // changer état afficheur LCD (interrupteur incrementiel)
//    timboupre = timbouact ;
//  }
//}
//_____________________________________________________________________________
void ctrlpwd1(EthernetClient client) {     // controle du mot de passe de premier niveau
  #if TRACES
    Serial.println("Controle mot de passe niveau 1 : ");
  #endif
  comspeciales(client);
  if(readString.indexOf("?password=") >0) {
    for (int i=0;  i < readString.length();  i++) {
      if (readString[i] == '=') {
        passwd = readString.substring(i+1,i+7);   //  lire mot de passe en 6 caractères
        break;
      }
    }
    if (passwd == motpas1)  {    // contrôle du mot de passe tapé avec le mot de référence
      pwdok1 = true;
      deconn = false;
      heurdeb = millis();   // stockage heure debut de validité
      #if TRACES
        Serial.print("Password 1 OK : ");
        Serial.println(passwd);
      #endif
    }
  }
```

```
}
//_____________________________________________________________________________
void comspeciales(EthernetClient client) {
  if(readString.indexOf("?RzFcom") >0) {   // demande d'effacement du fichier HIACTIV
    nivosecu = 2;
    pwdok2 = false;
    indeff = true;      // indic effacement fichier histo des commandes
    afficomm(client);
    affich1(client);    // affichage demande du mot de passe niveau 2
  }
  if(readString.indexOf("?Arcme") >0) {   // demande archivage mesures mois précéd.
    nivosecu = 2;
    pwdok2 = false;
    indarch = true;      // indic archivage mesures mois précédents
    afficomm(client);
    affich1(client);    // affichage demande du mot de passe niveau 2
  }
  if(readString.indexOf("?Coff") >0) {  // demande de fermeture de la page commandes
    pwdok1 = false;
    pwdok2 = false;    // si mot de passe erroné..
    nivosecu = 1;      // retour au premier niveau securité
  }
}
//_____________________________________________________________________________
void ctrlpwd2(EthernetClient client) {     // controle du mot de passe de second niveau
  #if TRACES
    Serial.println("Controle mot de passe niveau 2 : ");
  #endif
  comspeciales(client);
  if(readString.indexOf("?password=") >0) {
    for (int i=0;  i < readString.length();  i++) {
      if (readString[i] == '=') {
        passwd = readString.substring(i+1,i+7);   //  lire mot de passe en 6 caractères
        break;
      }
    }
    if (passwd == motpas2)  {    // contrôle du mot de passe tapé avec le mot de référence
      pwdok2 = true;
      #if TRACES
        Serial.print("Password 2 OK : ");
        Serial.println(passwd);
      #endif
    }
    else {
      pwdok2 = false;    // si mot de passe erroné..
      nivosecu = 1;      // retour au premier niveau securité
    }
  }
}
//_____________________________________________________________________________
void controlduree() {     // contrôle durée du mot de passe
        if (pwdok1 == true) {
```

```
          duree = millis() - heurdeb;
          if (duree > 60000) {   // test si validité mot de passe expirée (1 minute)
            pwdok1 = false;
            pwdok2 = false;
            deconn = true;
            nivosecu = 1;      // retour au premier niveau securité
          }
        }
}
//_____________________________________________________________________________
void action1() {   // allumer ou couper les relais selon commandes reçues
        command = false;
        if(readString.indexOf("?R1") >0) {
          command = true;
          if (rel1 == 0)   rel1 = 1;
          else             rel1 = 0;
        }
        if(readString.indexOf("?R2") >0) {
          command = true;
          if (rel2 == 0)   rel2 = 1;
          else             rel2 = 0;
        }
       if(readString.indexOf("?R3") >0) {
          command = true;
          if (rel3 == 0)   rel3 = 1;
          else             rel3 = 0;
        }
        if(readString.indexOf("?R4") >0) {
          command = true;
          if (rel4 == 0)   rel4 = 1;
          else             rel4 = 0;
        }
        if(readString.indexOf("?Himmc") >0)    indhimmc = true;  // demande aff.histo mesures mois
en cours
        if(readString.indexOf("?Himto") >0)    indhimto = true;  // demande aff.histo toutes les
mesures
        if(readString.indexOf("?Hisco") >0)    indhisco = true;  // demande aff.histo des
commmandes
        if (command == true) {  // si un ordre a été donné..
          activrel();
          enrtraceact();      // enregistrer trace de la commande dans fichier HIACTIV
        }
}
//_____________________________________________________________________________
void activrel() {      // actionner les relais selon leur état
  if (rel1 == 1) {
    digitalWrite(out1, HIGH);   // allumer relais 1
    radio.send(262227, 24);     // envoi code ON pour allumage prise radiocommandée "C"
   } else {
      digitalWrite(out1, LOW);    // couper relais 1
      radio.send(262236, 24);     // envoi code OFF pour allumage prise radiocommandée "C"
    }
```

```
  if (rel2 == 1) {
    digitalWrite(out2, HIGH);    // allumer relais 2
    radio.send(4194387, 24);     // envoi code ON pour allumage prise radiocommandée "B"
   } else {
       digitalWrite(out2, LOW);     // couper relais 2
       radio.send(4194396, 24);     // envoi code OFF pour allumage prise radiocommandée "B"
     }
  if (rel3 == 1) {
    digitalWrite(out3, HIGH);  // allumer relais 3
    radio.send(83, 24);     // envoi code ON pour allumage prise radiocommandée "A"
   } else {
       digitalWrite(out3, LOW);  // couper relais 3
       radio.send(92, 24);     // envoi code OFF pour allumage prise radiocommandée "A"
     }
  if (rel4 == 1)  digitalWrite(out4, HIGH);   // allumer relais 4
  else digitalWrite(out4, LOW);       // couper relais 4
}
//_____________________________________________________________________________
void action2(EthernetClient client) {   // executer commandes de niveau 2
  if (indeff == true) {
    #if TRACES
      Serial.println("action niveau 2 :  RAZ fichier commandes ");
    #endif
    effacehicom();  // effacement du fichier trace des commandes
    nivosecu =1;   // retour au premier niveau securité apres action niveau 2
    actio2 = true;    // indic commande de niveau 2 effectuée
    indeff = false;   // annuler la demande apres son exécution
  }
    if (indarch == true) {
    #if TRACES
      Serial.println("action niveau 2 :  archivage fichier mesures mois precedents");
    #endif
    archivmesur();   // archivage fichier mesures mois precedents
    nivosecu =1;   // retour au premier niveau securité apres action niveau 2
    actio2 = true;   // indic commande de niveau 2 effectuée
    indarch = false;   // annuler la demande apres son exécution
  }
}
//_____________________________________________________________________________
void mesuredht22() {    // taux humidité et temp intérieure sur le DHT22
    #if TRACES
      Serial.println("fonction : mesuredht22");
    #endif
    codret = DHT.read22(DHT22PIN);
    switch (codret) {
      case DHTLIB_OK:
        humip = DHT.humidity;
        tempint = DHT.temperature;
        #if TRACES
          Serial.print("humidite = ");
          Serial.print(humip);
          Serial.print("  temperature = ");
```

```
        Serial.println(tempint);
       #endif
       break;
     default:
       humip = 0;
       #if TRACES
         Serial.println("Erreur lecture capteur DHT22");
       #endif
       break;
   }
}
//_____________________________________________________________________________
String zerogauch2(int nombre) {   // ajouter zéros non significatifs si nombre nn < taille voulue
  String valnch = "";
  if (nombre < 10)   { valnch = "0"; }
  valnch += String(nombre, DEC);
  return valnch;      // renvoi le nombre en 2 chiffres quelle que soit sa valeur
}
//_____________________________________________________________________________
String zerogauch4(int nombre) {   // ajouter zéros non significatifs si nombre nnnn < taille voulue
  String valnch = "";
  if (nombre < 1000) { valnch = "0"; }
  if (nombre < 100)  { valnch = "00"; }
  if (nombre < 10)   { valnch = "000"; }
  valnch += String(nombre, DEC);
  return valnch;      // renvoi le nombre en 4 chiffres quelle que soit sa valeur
}
//_____________________________________________________________________________
void affichlcd() {     // affichage données sur afficheur LCD
  if (digitalRead(swlcd1)) afflcd = 1;
  else
    if (digitalRead(swlcd2)) afflcd = 2;
    else
      if (digitalRead(swlcd3)) afflcd = 3;
      else afflcd = afflcdpre = 0;  // pas d'affichage LCD
  if (afflcd > 0) {      // test si switch d'affichage activé
    if (afflcd != afflcdpre) {    // si changement d'état
      lcd.clear();            // reinitialiser affichage LCD
      lcd.setBacklight(LOW);  // activer le rétroeclairage
      #if TRACES
        Serial.print("activer afficheur LCD  etat : ");
        Serial.println(afflcd);
      #endif
      afflcdpre = afflcd;
    }
  }
  else {
    lcd.setBacklight(HIGH);  // éteindre le rétroeclairage
    lcd.clear();
    #if TRACES
      if (afflcd != afflcdpre)  Serial.println("couper afficheur LCD");
    #endif
```

```
}
if (afflcd == 1) {          // afficher données premier affichage
  if (millis() - milprelcd > 500) {    // temporiser affichage LCD 2 fois par seconde
    milprelcd = millis();
    DateTime now = RTC.now();
    lcd.setCursor (0,0); // curseur LCD en ligne 1
    if (now.day() < 10)  lcd.print("0");
    lcd.print(now.day(), DEC);
    lcd.print('/');
    if (now.month() < 10)  lcd.print("0");
    lcd.print(now.month(), DEC);
    lcd.print('/');
    lcd.print(now.year()-2000, DEC);   // afficher année en 2 chiffres
    lcd.setCursor (0,1);     // curseur LCD en ligne 2
    lcd.print('        ');
    lcd.setCursor (0,1);
    if (now.hour() < 10)  lcd.print("0");
    lcd.print(now.hour(), DEC);
    lcd.print(':');
    if (now.minute() < 10)  lcd.print("0");
    lcd.print(now.minute(), DEC);
    lcd.print(':');
    if (now.second() < 10)  lcd.print("0");
    lcd.print(now.second(), DEC);
    lcd.setCursor (9,0);
    lcd.print("Ti:");
    dtostrf(tempint,2,1,tempchar);   // conversion temp. int. de float en char
    lcd.print(tempchar[0]);         // dizaines de degrés
    lcd.print(tempchar[1]);         // unités
    lcd.print(char(223));           // signe '°' à la place de la virgule
    lcd.print(tempchar[3]);         // decimales
    lcd.setCursor (9,1);
    lcd.print("Pr:");
    lcd.print(presact);       // pression barom. actuelle
  }
}
if (afflcd == 2) {          // afficher données second affichage
  if (millis() - milprelcd > 500) {    // temporiser affichage LCD 2 fois par seconde
    milprelcd = millis();
    lcd.setCursor (0,0);   // curseur LCD en ligne 1
    int hsess = millis() / 3600000;
    int msess = ((millis() % 3600000) / 60000);
    lcd.print("ses:");
    lcd.print(hsess);     // durée de la session (depuis le reset) en heures....
    lcd.print("h");
    lcd.print(msess);     // ....et minutes
    lcd.print("m");
    lcd.setCursor (11,0);
    lcd.print("H:");
    lcd.print(humip);      // humidité
    lcd.print(char(37));
    lcd.setCursor (0,1);
```

```
      lcd.print("Ram:");
      lcd.print(freeRam());
      lcd.setCursor (9,1);
      lcd.print("E:");
      dtostrf(tempext,5,1,tempchar);     // conversion float en char
      lcd.print(tempchar[0]);         // signe
      lcd.print(tempchar[1]);         // dizaines de degrés
      lcd.print(tempchar[2]);         // unités
      lcd.print(char(223));          // caract "°" à la place de la virgule
      lcd.print(tempchar[4]);         // decimale
    }
  }
  if (afflcd == 3) {          // afficher données troisième affichage
    if (millis() - milprelcd > 500) {    // temporiser affichage LCD 2 fois par seconde
      milprelcd = millis();
      lcd.setCursor (0,0);   // curseur LCD en ligne 1
      lcd.print("RelON:");
      if (rel1==1)  lcd.print("1"); else lcd.print(" ");     // affichage état des relais
      if (rel2==1)  lcd.print("2"); else lcd.print(" ");
      if (rel3==1)  lcd.print("3"); else lcd.print(" ");
      if (rel4==1)  lcd.print("4"); else lcd.print(" ");
      lcd.setCursor (0,1);
      lcd.print("NbConn:");
      lcd.print(nbconn);
    }
  }
}
//_____________________________________________________________________________
void memoeeprom(int d) {         // Memoriser un timbre dans Eprom (pos d sur 4 bytes)
      DateTime now = RTC.now();
      secondactu = now.unixtime();
      eeptimbre[0] = (int)((secondactu >> 24) & 0xFF) ;   // convertir long en un tableau de 4 bytes
      eeptimbre[1] = (int)((secondactu >> 16) & 0xFF) ;
      eeptimbre[2] = (int)((secondactu >> 8) & 0XFF);
      eeptimbre[3] = (int)((secondactu & 0XFF));
      for (int i=0; i<4; i++) {          // stocker timbre enregt dans Eeprom du DS1307 (pos d sur 4
bytes)
        int j = i+d;
        RTC.writenvram(j,eeptimbre[i]);
      }
      #if TRACES
        if (d == 0) {              // => enregt timbre stockage fich. MESURES.txt
          Serial.print("Enregt. timbre dern.stockage MESURES : ");
        }
        if (d == 4) {              // => enregt timbre de synchro serveur NTP
          Serial.print("Enregt. timbre dern. synchro serveur NTP : ");
        }
        if (d == 8) {              // => enregt timbre de test de la variation barométrique
          Serial.print("Enregt. timbre test variation pression : ");
        }
        Serial.println(secondactu);
      #endif
```

```
}
//
_____________________________________________________________________________________*/
int freeRam()  {     // calcule la quantité de mémoire SRAM libre
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
//_____________________________________________________________________________________
void affheurserie()  {    // affichage date et heure pour suivi sur port série
  #if TRACES
    DateTime now = RTC.now();
    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(' ');
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.print("  temps Posix : ");
    Serial.print(now.unixtime());
    Serial.println(" s");
  #endif
}
//_____________________________________________________________________________________
void ctrlvarbaro()  {    // contrôle de la variation de pression barométrique
  variapress();   // lecture des variations de pression depuis 48h
  if ((vario[1] < -6) && (vario[0] <= 0)) {   // conditions pour déclencher une alerte météo
    String message = "le ";
    DateTime now = RTC.now();
    message += now.day();
    message += "/";
    message += now.month();
    message += "/";
    message += now.year();
    message += " a ";
    message += now.hour();
    message += ":";
    message += now.minute();
    message += ":";
    message += now.second();
    message += "  ALERTE METEO : Chute de pression de ";
    message += vario[1];
    message += " Hpa en 6 heures";
    #if TRACES
      Serial.println(message);
    #endif
    char mess[message.length()+1];   // convertir le message de string en tableau de char
```

```
    message.toCharArray(mess, message.length()+1);
    envoitweet(mess);                // envoyer un tweet d'alerte
   }
}
//_____________________________________________________________________________
void envoitweet(char* mess)  {       // envoi d'un tweet d'alerte
    #if TRACES
      Serial.print("Envoi tweet alerte meteo. Message = ");
      Serial.println(mess);
    #endif
    if (twitter.post(mess)) {
      int status = twitter.wait();
      if (status == 200) {
        tweetok = 1;      // indic tweet envoyé OK
        #if TRACES
          Serial.println("Envoi tweet alerte meteo. OK");
        #endif
      }
      else {
        tweetok = 2;      // indic tweet envoyé KO (cas 1)
        #if TRACES
          Serial.print("Erreur envoi tweet alerte meteo. Code erreur : ");
          Serial.println(status);
        #endif
      }
    }
    else {
      tweetok = 3;      // indic tweet envoyé KO (cas 2)
      #if TRACES
        Serial.println("Erreur connexion pour envoi tweet alerte meteo.");
      #endif
    }
}
//_____________________________________________________________________________
void majhorloge() {   // actualisation de l'horloge par serveur NTP
    #if TRACES
      Serial.println("fonction : majhorloge");
    #endif
    DateTime now = RTC.now();
    if (now.month() > 3  && now.month() < 11)  decalhor = 2;
    else decalhor = 1;        // décalage horaire pour heure hiver (1) ou heure été (2)
    setSyncProvider(recupposix);
    if (majrtc == true) {   // on a eu accès au serveur NTP donc mettre à jour l'horloge RTC
      ann1 = year();
      moi1 = month();
      jou1 = day();
      heu1 = hour();
      min1 = minute();
      sec1 = second();
      RTC.adjust (DateTime(ann1,moi1,jou1,heu1,min1,sec1));    // synchro horloge RTC DS1307 sur
serveur NTP
      #if TRACES
```

```
        Serial.print("MaJ de l'horloge RTC : ");
        affheurserie();
      #endif
    }
}
//_____________________________________________________________________________
unsigned long int recupposix() {    // récup du temps posix sur serveur NTP
  majrtc = false;
  #if TRACES
    Serial.println("fonction : recupposix");
  #endif
  while (Udp.parsePacket() > 0) ;
  requetentp(serveurntp);
  uint32_t beginWait = millis();
  while (millis() - beginWait < 1500) {
    int size = Udp.parsePacket();
    if (size >= taillentp) {
      Udp.read(bufferntp, taillentp);  // lire le paquet reçu dans le buffer
      #if TRACES
        Serial.print("Acces serveur NTP : buffer recu = " );
        for (int i=0; i <= 48; i++){
          Serial.print(bufferntp[i]); }
        Serial.println(" ");
      #endif
      // conversion du timestamp (4 octets 40,41,42,43 (double mot) en un entier long
      duree1900 =  (unsigned long)bufferntp[40] << 24;
      duree1900 |= (unsigned long)bufferntp[41] << 16;
      duree1900 |= (unsigned long)bufferntp[42] << 8;
      duree1900 |= (unsigned long)bufferntp[43];
      memoeeprom(4);          // mémoriser dans eeprom date posix de synchro NTP (pos 4 à 7)
      tempsposix = (duree1900 - 2208988800UL + (decalhor * 3600L));   // stockage du temps posix
récupéré
      majrtc = true;      // indique que la synchro de l'horloge est à faire et le reset du systeme aussi)
      return (tempsposix);
    }
  }
  #if TRACES
    Serial.println("sans reponse du serveur NTP");
  #endif
  return (secondactu);    // renvoi du temps actuel sans MaJ de l'horloge
}
//_____________________________________________________________________________
void requetentp(IPAddress &address) {    // Initialisation et envoi de la requête NTP
  memset(bufferntp, 0, taillentp);  // raz buffer ntp
  bufferntp[0] = 0b11100011;    // LI, Version, Mode
  bufferntp[1] = 0;             // Stratum, or type of clock
  bufferntp[2] = 6;             // Polling Interval
  bufferntp[3] = 0xEC;          // Peer Clock Precision
  bufferntp[12]  = 49;
  bufferntp[13]  = 0x4E;
  bufferntp[14]  = 49;
  bufferntp[15]  = 52;
```

```
  Udp.beginPacket(address, 123);    // requetes NTP pour demander un timestamp (sur port externe
123)
  Udp.write(bufferntp,taillentp);
  Udp.endPacket();
}
//_____________________________________________________________________________________
void calcultimbre() {    // calcul du timbre pour horodatage enregistrements
    timbre = "";
    DateTime now = RTC.now();
    timbre += (now.year());
    timbre += (zerogauch2(now.month()));
    timbre += zerogauch2(now.day());
    timbre += zerogauch2(now.hour());
    timbre += zerogauch2(now.minute());
    timbre += zerogauch2(now.second());
    #if TRACES
      Serial.print("timbre : ");
      Serial.println(timbre);
    #endif
}
//_____________________________________________________________________________________
//_____________________________________________________________________________________
```

Sonde de température

```
/*****************************************************************
SONDE_TEMP_RADIO    v1.4
Envoi par radio de la mesure de la température (pour prog REL4YLC)
auth: Yves Le Chevalier  23/02/2015
config : Atmega328P-PU + MR003(TCN75A) en I2C + émetteur RF433 Mhz
NB : Alim. TNC75A de 2,7 à 5,5v.  Emetteur RF433Mhz 3,5 à 12v.   => alim 3,7v. OK
_____________________________________________________________________
Affectation des entrées-sorties :
9   Data à envoyer sur émetteur radio
A4  (ie3) I2C - SDA
A5  (ie2) I2C - SCL
_____________________________________________________________________
__
v1_0 : prototype sur carte Nano avec envoi mesure température chaque seconde
v1_1 : mise en veille et envoi température toutes les 8 sec
v1_2 : conso réduite par mise en veille totale et envoi température chaque minute
v1_3 : calcul température moyenne de 10 relevés successifs (lissage écarts)
v1_4 : adapter prog. pour atmega328P-PU en standalone   (0,115 µa en sommeil)
_____________________________________________________________________
___*/

// sonde TCN75A températ I2C :
#include <Wire.h>
float temp = 0;
byte address = 0x48;

// emetteur radio :
#include <RCSwitch.h>
RCSwitch radio = RCSwitch();
int sonde = 2;  // alimenter sonde temp par pin 2

#include <avr/wdt.h>
#include <avr/sleep.h>
#include <avr/power.h>
// watchdog interrupt :
ISR (WDT_vect) {
  wdt_disable();  // desactiver le watchdog
}
void myWatchdogEnable() {
  MCUSR = 0;  // effacer divers "reset" flags
  WDTCSR = bit (WDCE) | bit (WDE);  // permet modif et desactivation du reset
  // activer le mode interruption pour une durée donnée
  WDTCSR = bit (WDIE) | bit (WDP3) | bit (WDP0);    // set WDIE, et délais de 8 sec.
  wdt_reset();  // pat the dog
  ADCSRA = 0;   // desactiver l'ADC
  set_sleep_mode (SLEEP_MODE_PWR_DOWN);  // paramétrer le mode de sommeil
  sleep_enable();
  MCUCR = bit (BODS) | bit (BODSE);  // couper brown-out détection (bas voltage alim)
  MCUCR = bit (BODS);
  sleep_cpu ();
```

```
  sleep_disable();  // désactiver sommeil par precaution
}


void setup() {
  pinMode(sonde, OUTPUT);
  digitalWrite(sonde, HIGH);  // brancher alim du TCN75A
  Wire.begin();
  Wire.beginTransmission(address);  // réglages du TCN75A :
  Wire.write(0x01);   // adresser le registre de configuration
  Wire.write(0x60);   // pour régler la résolution à 12 bits
  Wire.endTransmission();
  Wire.beginTransmission(address);
  Wire.write(0x00);  // ré-adresser le registre de lecture de la température
  Wire.endTransmission();   // fin réglages sur TCN75A
  digitalWrite(sonde, LOW);  // couper alim du TCN75A
  radio.enableTransmit(9);  // data emetteur radio sur pin pwm 9
  radio.setRepeatTransmit(10);  // nb d'envois de la valeur a chaque émission
  power_adc_disable();     // désactiver convert. Analog/Digital
  power_spi_disable();     // désactiver bus SPI
  power_twi_disable();   // désactiver bus I2C
  power_usart0_disable();  // désactiver port USB
  power_timer0_disable();  // désactiver timer0 (ne pas désactiver si milli() ou delay())
  power_timer1_disable();  // désactiver timer1
  power_timer2_disable();  // désactiver timer2
}

void loop() {
    digitalWrite(sonde, HIGH);  // alimenter sonde temp.
    power_twi_enable();   // activer bus I2C
    power_timer0_enable();
    delay(300);
    power_timer0_disable();
    temp = calctemp()*100;   // mesure température  (x100 pour envoi radio)
    if (temp < 0)     temp = abs(temp) + 10000;   // caratériser tempér. négative
    temp = temp + 5555555;  // pour ne pas interférer sur les codes radio des relais
    radio.send(temp, 24);
    digitalWrite(sonde, LOW);  // couper alimentation sonde temp.
    power_twi_disable();   // désactiver bus I2C
    // mises en veille répétitives pour atteindre 1 minute de veille totale
    for (int i = 0; i < 7; i++)  // 8x 8 sec = veille pdt 60 sec
    myWatchdogEnable ();
}
//
```

```
_
  float mesuretemp()  {  // lire et convertir température sur MR312
  Wire.requestFrom(address,byte(2));
  int tempreg = Wire.read();
  tempreg= tempreg << 8;
  tempreg |= Wire.read();
  tempreg = tempreg >> 4;
```

```
  float temper =( float ) tempreg / 16;  // calcul température en degrés
  return(temper);
}
//
```

```
_
float calctemp() {    // recuperer temp  (sur 10 mesures)
  float valtot = 0;
  int i;
  for (i=0; i<10; i++) {
    valtot = valtot + mesuretemp();
  }
  float valmoy = valtot / i;    // calcul moyenne des mesures
  return (valmoy + 0.9);     // ajustage temp mesurée/temp constatée (en °c)
}
//
**********************************************************************************
```