```
/*    Supervision de cloture electure electrique
 *
 *   (c)2021  Henry-Pascal ELDIn  Eldin.net
 *
 *    10 septembre 2021
 *
 */

#include <GPRS_Shield_Arduino.h>
#include <SoftwareSerial.h>

#include <time.h>


//define VERSION "1.0a"    // 10 septembre 2021    mise en place
//define VERSION "1.0b"    // 15 septembre 2021    ajout date
//define VERSION "1.0c"    // 19/09/2021     lib LayadCircuits_SalengGSM
//define VERSION "1.0e"      //20/09/2021    lib GPRS_Shield_Arduino
//define VERSION "1.0f"     // 22/09/2021     tests operationnels   mode debug
//define VERSION "1.0g"      //23/09/2021      ajout compteur mode debug
//#define VERSION "1.0h"        // 26/09/2021    correction bug  mode DEBUG
                              //                 SEUIL a 25
//#define VERSION "1.0i"        // optimisation  ajout code reboot arduino hard
//#define VERSION "1.0j"        // 06/10/2021 optimisation  correction bug envoi
sms multiple
//#define VERSION "1.0k"        // 08/10/2021 optimisation  correction bug envoi
entete sms
//#define VERSION "1.0l"          // 19/10/2021 suppression des fonctions non
utilisés de la lib GPRS_Shield, modif seuil de flash, ajout sms aide
//#define VERSION "1.0m"          //23/10/2021  pb compteur flash dans msg, sup
sms aide, ajout qualite gsm debug
//#define VERSION "1.0n"            //23/10/2021  pb photo resistance
#define VERSION "1.0o"            // 23/10/2021 pb compteur flash vu

#define LIEU "Le_Lieu";  // site

// config SIM900
#define PIN_TX    7
#define PIN_RX    8
#define PIN_POWER 9
#define BAUDRATE  9600

#define MESSAGE_DMD_AIDE "#00#"
#define MESSAGE_DMD_INFO "#01#"
#define MESSAGE_1_SMS_MINUTE "#97#"
#define MESSAGE_1_SMS_HEURE  "#98#"
#define MESSAGE_1_SMS_24_HEURES "#99#"
#define MESSAGE_RAZ_DMD_SMS   "#96#"
#define MESSAGE_VALID_CLOTURE "#33#"
#define MESSAGE_VALID_BATTERIE "#34#"
#define MESSAGE_DMD_REBOOT  "#4321#"
#define MESSAGE_DMD_DEBUG "#22#"

#define MESSAGE_DMD_SEUIL_DEF "#40#"
#define MESSAGE_DMD_SEUIL_INC "#41#"
#define MESSAGE_DMD_SEUIL_DEC "#42#"


#define MESSAGE_AIDE_0 "#00#  Aide\n#01# Status\n#22# Compteurs\n#33# Valid
Alarm Cloture\n#34# Valid Alarm Batterie\n"
#define MESSAGE_AIDE_1 "#40# Seuil flash def\n #41# Seuil flash +1\n#42# Seuil
flash -1\n"
#define MESSAGE_AIDE_2 "#96# raz Dmd SMS\n#97# 1 sms /mn\n#98# 1 sms /h\n#99# 1
sms /24h\n"
```

```
#define MESSAGE_PB_PHOTO_RESIS "Pb interne resistance photo "
#define MESSAGE_DEBUG1 "Debug 1 sms par 24h\n"
#define MESSAGE_DEBUG2 "Debug 1 sms toutes les heures\n"
#define MESSAGE_DEBUG3 "Debug 1 sms toutes les minutes\n"
#define MESSAGE_RAZ_DEBUG "Debug raz des envois sms\n"

#define MSG_VALID_CLOTURE "Validation alarme Cloture\n"
#define MSG_VALID_BATTERIE "Validation alarme Batterie\n"

#define MSG_REBOOT "Reboot du systeme\n"




// pour emission SMS
const char * Correspondant[]={ "+33601020304"};
//,"+33602030405" };

String Correspondant_Debug="";
// pour reception SMS

#define MESSAGE_LENGTH 160
char message[MESSAGE_LENGTH];
unsigned short int messageIndex = 0;
char phone[16];


/* gestion ddes boucles temps   */
unsigned short int CentiemeSeconde=0;   // incremente les centieme de secondes
unsigned short int Seconde=0;           // incremente les secondes
unsigned short int Minute=0;            // incremente les minutes
unsigned short int DixiemeSeconde=0;    // incremente les dixiemes de secondes
unsigned short int Heure=0;             // increment les heures
unsigned short int Jour=0;              // increment des jours


/* gestion du flash de la cloture */
unsigned short int FlashVu=0;             //incrementer si le flash est vu
unsigned short int lightPin=0;            // Pin de la resistance photo
bool Alarme_Cloture=false;                // true ou false  absence de tension
unsigned short int ValeurFlash=0;         // valeur de l'intensité du flash lu
String m="";                             // chaine a afficher
#define SEUIL  15                         // Nb minimum de vue du flash par
minute par defaut
#define FLASH  80                         // valeur minimum de l'intensité du
flash pour le  valider

unsigned short int Seuil=0;                // seuil de flash vu dynamique
bool Valid_Cloture=false;                 // validation de l'alarme Cloture



/* gestion de la tension de la batterie   */

unsigned short int pinTension=1;              // pin pour lire la tension de la
batterie
unsigned short int val=0;                     // valeur de la tension lue sur
la pin
float TENSION=0;                              // valeur de la tension de la
batterie
```

```
bool Alarme_Batterie=false;                    // true ou  false tension < 12v
#define BATTERIE  12                           // tension de référence pour la
batterie  12,00 volts par défaut
#define COEF_TENSION 19.57                     //  coefficient de calcul de la
tension lue
                                               //  =  valeur_lue_sur_la_pin /
tension_mesurée_sur_la_batterie
                                               //  au meme moment
float  coef_tension=COEF_TENSION;
bool Valid_Batterie=false;                     // validation de l'alarme
Batterie


// Gestion Date heure
char DateTime[24];                             // au format   YY/MM/DD
HH:MM:SS+08"
time_t timestamp_debut;                        // timestamp de l'heure de
lancement du programme
time_t timestamp_courant;                      // timestamp de l'heure en cours
time_t timestamp_ecoule;                       // timestamps du temps ecoule
depuis le lancement
struct tm myDate;                              // structure tm pour travail sur
la date

//qualité du signal
unsigned short int QUALITE=0;


// mode debug
bool debug1=false;       //  #99#  un sms toutes les 24h
bool debug2=false;       //  #98#  un sms toutes les heures
bool debug3=false;       //  #97#  un sms toutes les minutes


// compteurs
unsigned short int compteur_sms_emis=0;      // compteur nb sms emis
unsigned short int compteur_sms_recu=0;      // compteur nb sms recus
unsigned short int compteur_alarme_cloture=0;  // compteur nb d'alarme de
cloture
unsigned short int compteur_alarme_batterie=0; // compteur nb d'alarme de
batterie

#define NB_DIXIEME_SECONDE  20                  // nombre de boucle dixieme de
secondes
#define NB_SECONDE          58                  // nombre de boucle seconde dans
une minute
#define NB_MINUTE           60                  // nombre de minute dans un
boucle heure
#define NB_HEURE            24                  // nombre de boucle heure dans
un jour

// heures d'envoi systematique d'un sms d'info
// par défaut à 8h et 17h
#define HEURE_ENVOI_SMS_1   8
#define HEURE_ENVOI_SMS_2   17



GPRS gprs(PIN_TX, PIN_RX, BAUDRATE); //init module SIM900  RX,TX,PWR,BaudRate



void setup() {
```

```
    Serial.begin(9600);                    // pour debug
    Serial.print("Deb Setup");
    Serial.println(VERSION);

    while ( ! gprs.checkPowerUp()) {         // test si module sim900 allumé
        gprs.powerUpDown(PIN_POWER);      // allume module sim900
    }



    while ( !gprs.isNetworkRegistered()) {         // test si enregistré sur le
réseau GSM
        delay(1000);
    }

    while (  !gprs.init())  {                // test si le module sim900 est
operationnel
        Serial.print("init error\r\n");
        delay(1000);
    }

    delay(3000);


    gprs.getDateTime(DateTime);           // recup la date depuis sim900
    Init_Heure_Debut();                   // stocke l'heure dans un timestamp
    gprs.getSignalStrength(QUALITE);      // qualité du signal GSM




    Lecture_Tension();                    // lecture de  la tension de la
batterie
                                          // la premiere lecture est pas
toujours bonne
    TENSION=Lecture_Tension();           // lecture de  la tension de la
batterie
    Envoi_SMS(Genere_Info());                          // envoi sms  de demarage
    Serial.println("Setup OK ");
    //Serial.println(VERSION);
}

void loop() {

     delay(50);                     // attente 50 millisecondes


     Lire_Flash();                // voir si le flash est present

     DixiemeSeconde++;          // incremente
        // boucle seconde
        if (DixiemeSeconde == NB_DIXIEME_SECONDE ) {     // si seconde atteinte
            DixiemeSeconde=0;        // les dixiemes a zéro
            Seconde++;           // incremente seconde
        }

         // boucle minute
         if (Seconde == NB_SECONDE ) {       // attente de 60 secondes
```

```
        Seconde=0;              //  seconde à zéro
        Minute++;               // incremente minute
        /*
        Serial.print("FlashVu = ");
        Serial.print(FlashVu);
        Serial.print(" - ");
        Serial.println(Minute);
        */

         TENSION=Lecture_Tension();                  // lecture de la tension de
la batterie

         if ( (FlashVu < SEUIL) && (Alarme_Cloture == false) ) {  // pas vu le
flash et pas d'alarme
            Alarme_Cloture=true;                                  // passe a
ALARME
            compteur_alarme_cloture++;                            // increment
le compteur
            Envoi_SMS(Genere_Info());                             // envoi un
sms
         }             // fin pas vu le flash et pas d'alarme


         if ( FlashVu > SEUIL && Alarme_Cloture == true )  {   // le flash a été
vu et une alarme dans la minute avant
             Alarme_Cloture=false;       // passe a false
             Valid_Cloture=false;
             Envoi_SMS(Genere_Info());    // retablissement
         }           //  fin le flash a été vu et une alarme dans la minute
avant


         if ( FlashVu > SEUIL && Alarme_Cloture == false )  {   // pas d'alarme
raz des valeurs
             Alarme_Cloture=false;       // passe à false
             Valid_Cloture=false;
          }         // fin   pas d'alarme raz des valaurs

         if ( Alarme_Cloture == true && Valid_Cloture == false ) {    // il y a
une alarme cloture non validée
            Envoi_SMS(Genere_Info());                                 // on envoi
un SMS
        }


        if ( FlashVu > (SEUIL*10)  ) {                // si un soucis avec la
photo resistance
            Serial.print(MESSAGE_PB_PHOTO_RESIS);
            Serial.println(FlashVu);
            Envoi_SMS(Genere_Msg_Erreur(MESSAGE_PB_PHOTO_RESIS));
         }


        if ( debug3 == true) {                      // mode debug 3 suivi toutes
les minutes des infos
            Envoi_SMS_Debug(Genere_Info(),Correspondant_Debug);
        }
        if ( Alarme_Batterie==  true  && Valid_Batterie == false)  {  // alarme
de batterie non validée
            Envoi_SMS(Genere_Info());
        }


        Lire_SMS();                     // regarde si un sms arrivé et on le
```

```
traite


         // traitement toutes les heures

      // le nb d'heures est arrivée au maxi   dans la boucle jour
      if ( Heure == NB_HEURE )  {
            Heure=0;
            Jour++;
            if ( (debug1 == true ) ) {      //  si mode debug 1
              Envoi_SMS_Debug(Genere_Info(),Correspondant_Debug);
            }
        }

     // le nb de minute est au maxi dans la boucle heure
      if (Minute == NB_MINUTE ) {       // attente de 60 minutes
        Minute=0;              //  minutes à zéro
        Heure++;               // incremente heure
        if ( (debug2== true) || (Alarme_Cloture == true) || ( Alarme_Batterie ==
true) ) {   // si mode debug 2 ou alarme cloture ou alarme batterie
          Envoi_SMS_Debug(Genere_Info(),Correspondant_Debug);
        }



        // on envoi un sms d'info le matin et le soir
        if ( Test_Horaire_Envoi_SMS() == true  ) {
              Envoi_SMS(Genere_Info());
        }
      } // fin boucle minute

       FlashVu=0;                                  // raz du nb de flash




    }                          // fin boucle minute



}//  fin loop



void(* resetFunc) (void) = 0;         //declare reset function at address 0


/* lecture de la tenson la batterie   */
float  Lecture_Tension() {
    float t;
    Serial.print("Tension ");

    val = analogRead(pinTension);           // lecture du pin
    t=(val*coef_tension)/1000;              // calcul de la tension réelle
    Serial.print(val);
    Serial.print("/");
    Serial.println(t);

    if ( ( t <= BATTERIE ) && ( Alarme_Batterie == false )) {               //
si la tension est inferieure a la tension de ref
      Alarme_Batterie=true;              //  alors alarme
      compteur_alarme_batterie++;
```

```
    }
    if ( ( t > BATTERIE ) && ( Alarme_Batterie == true )  ) {   // si tension
est sup à la tension de ref
      Alarme_Batterie=false;                      // et alarme en court alors
retablissement ok
      Valid_Batterie=false;

    }
   return t;
}



/* genere le texte du SMS   */
/* attention maxi 140 caracteres */
  String Genere_Info(){                          // génération du message d'info
  String mn="";
  gprs.getSignalStrength(QUALITE);            // lecture qualité signal sim900
  mn=Entete();
  mn+="Bat: ";
  if ( TENSION >= BATTERIE )
      mn+="OK ";
  else
      mn+="HS ";
  mn+=TENSION;
  mn+="v";
  if ( Valid_Batterie == true ){
    mn+=" V";
  }
  mn+="\n";
  mn+="Cloture: ";
  if (Alarme_Cloture == true)
      mn+="HS ";
  else
      mn+="OK ";
  mn+=FlashVu;
  mn+="/mn";
  if ( Valid_Cloture == true ){
    mn+=" V";
  }
  mn+="\n";
  mn+="Qual GSM: ";
  int Q=(100-(15*QUALITE));
  mn+=Q;
  mn+="%\n";
  return mn;
}

// genere le message d'erreur
String Genere_Msg_Erreur(String err){
    String m1="";
    m1=Entete();
    m1+="\n";
    m1+=err;
    m1+="\n";
  return m1;
}

// Genere le message de debug
String Genere_Msg_Debug(){
  String m1="";
  //m1=Entete();
  // valeur des flags de debug
  m1="Dbg: ";
```

```
  m1+=debug1;
  m1+="/";
  m1+=debug2;
  m1+="/";
  m1+=debug3;
  m1+="\n";
  m1+="Dest:";
  m1+=Correspondant_Debug;
  m1+="\n";
  // valeurs de compteurs nb sms et alarmes
  m1+="Cpt: ";
  m1+=compteur_sms_emis;
  m1+="/";
  m1+=compteur_sms_recu;
  m1+="/";
  m1+=compteur_alarme_cloture;
  m1+="/";
  m1+=compteur_alarme_batterie;
  m1+="\n";
  // données sur la batterie
  m1+="Bat:";
  m1+=BATTERIE;
  m1+="/";
  m1+=coef_tension;
  m1+="/";
  m1+=val;
  m1+="/";
  m1+=TENSION;
  m1+="v";
  m1+="\n";
  // données sur la cloture
  m1+="Cloture:";
  m1+=SEUIL;
  m1+="/";
  m1+=FLASH;
  m1+="/";
  m1+=FlashVu;
  m1+="\n";
  // données qualité GSM
  gprs.getSignalStrength(QUALITE);
  m1+="Qual GSM: ";
  m1+=QUALITE;
  int Q=(100-(15*QUALITE));
  m1+="/";
  m1+=Q;
  m1+="%";
  m1+="\n";
  //Serial.println(m1);
  return m1;
}



/* envoi le SMS d'init  */
void Envoi_SMS( String str){

  //Serial.println("Envoi_SMS");
  //Serial.println(str);
  /*
  int lmsg=strlen(str.c_str());
  Serial.print("Longeur msg ");
  Serial.println(lmsg);
  */
  for (int i=0;i<(sizeof(Correspondant)/2); i++){  // on divise par 2 pointeurs
```

```
systeme 64bits
    /*
      Serial.print(sizeof(Correspondant));
      Serial.print(" -- ");
      Serial.print(i);
      Serial.print("->");
      Serial.println(Correspondant[i]);
      */
      compteur_sms_emis++;

      if (gprs.sendSMS(Correspondant[i],str.c_str() )) { //define phone number
and text
        Serial.println("SMS ok");
        } else {
        Serial.println("SMS erreur");
        }

    }   // fin des correspondants

} // fin Envoi_SMS


/* envoi le SMS de debug  */
void Envoi_SMS_Debug( String str ,String Dest){

  //Serial.println("Envoi_SMS Debug");
  Serial.println(str);
  /*
  int lmsg=strlen(m.c_str());
  Serial.print("L msg ");
  Serial.println(lmsg);
  */
  compteur_sms_emis++;

  if (gprs.sendSMS(Dest.c_str(),str.c_str() )) { //define phone number and text
        Serial.println("SMS Dbg ok");
        } else {
        Serial.println("SMS Dbg erreur");
        }

}   // fin envoi le SMS de debug


/* gestion de la lesture des SMS recus  */
void Lire_SMS() {
    //    Serial.println("Lire_SMS");

        messageIndex = gprs.isSMSunread();      // regarde si des sms ont pas
été lu


        if (messageIndex > 0) {                 //  un sms a été trouvé
            compteur_sms_recu++;

            gprs.readSMS(messageIndex, message, MESSAGE_LENGTH, phone,
DateTime);  // lit le SMS
            gprs.deleteSMS(messageIndex);
// on l'efface de la carte sim

            // on regarde si l'appelant est dans la liste des numéros autorisés
            for (int i=0;i<(sizeof(Correspondant)/2);i++){
                    if ((NULL != strstr(phone, Correspondant[i] ))) {       //
le numéro est autorisé
                            /*
```

```
                            //  demande aide
                           if ((NULL != strstr(message,MESSAGE_DMD_AIDE ))) {
                                     //Serial.println("Dmd aide");
                                     Envoi_SMS(MESSAGE_AIDE_0);
                                     Envoi_SMS(MESSAGE_AIDE_1);
                                     Envoi_SMS(MESSAGE_AIDE_2);
                             }
                             */

                             // demande de status
                             if ((NULL != strstr(message,MESSAGE_DMD_INFO ))) {
                                     //Serial.println("Dmd info");
                                     Envoi_SMS(Genere_Info());
                             }
                             /*
                             // un sms toutes les 24h
                             if ((NULL !=
strstr(message,MESSAGE_1_SMS_24_HEURES ))) {
                                 Correspondant_Debug=phone;
                                 //Serial.println("debug1");

Envoi_SMS_Debug(MESSAGE_DEBUG1,Correspondant_Debug);
                                 debug1=true;
                             }
                             */
                             // un sms toutes les heures
                             if ((NULL != strstr(message,MESSAGE_1_SMS_HEURE )))
{
                                 Correspondant_Debug=phone;
                                 //Serial.println("debug2");

Envoi_SMS_Debug(MESSAGE_DEBUG2,Correspondant_Debug);
                                 debug2=true;
                             }
                             // un sms toutes les minutes
                             if ((NULL != strstr(message,MESSAGE_1_SMS_MINUTE )))
{
                                   Correspondant_Debug=phone;

Envoi_SMS_Debug(MESSAGE_DEBUG3,Correspondant_Debug);
                                   //Serial.println("debug3");
                                   debug3=true;
                             }
                             // raz des modes d'envoi
                             if ((NULL != strstr(message,MESSAGE_RAZ_DMD_SMS )))
{
                                   Correspondant_Debug=phone;

Envoi_SMS_Debug(MESSAGE_RAZ_DEBUG,Correspondant_Debug);
                                   //Serial.println("RAZ debug");
                                   debug1=false;
                                   debug2=false;
                                   debug3=false;
                                   Correspondant_Debug="";
                               }

                               // Validation de l'alarme Cloture
                               if ((NULL !=
strstr(message,MESSAGE_VALID_CLOTURE ))) {
                                   //Serial.println("V Alarme Cloture" );
                                   Envoi_SMS_Debug(MSG_VALID_CLOTURE,phone);
                                   Valid_Cloture=true;
                               }
```

```
                                // Validation de l'alarme Batterie
                                if ((NULL != strstr(message,MESSAGE_VALID_BATTERIE
))) {
                                        //Serial.println("V Alarme Batterie" );
                                        Envoi_SMS_Debug(MSG_VALID_BATTERIE,phone);
                                        Valid_Batterie=true;
                                 }

                                 // msg de debug
                                 if ((NULL != strstr(message,MESSAGE_DMD_DEBUG )))
{
                                            //Serial.println("Msg debug" );
                                            String m1="";
                                            m1=Genere_Msg_Debug();
                                            Envoi_SMS_Debug(m1,phone);
                                 }

                                 // msg de seuil defaut
                                 if ((NULL != strstr(message,MESSAGE_DMD_SEUIL_DEF
))) {
                                            //Serial.println("Msg seuil def" );
                                            Seuil=SEUIL;
                                            String m1="";
                                            m1=Genere_Msg_Debug();
                                            Envoi_SMS_Debug(m1,phone);
                                 }

                                 // msg de  increment de seuil
                                 if ((NULL != strstr(message,MESSAGE_DMD_SEUIL_INC
))) {
                                            //Serial.println("Msg seuil inc" );
                                            Seuil++;
                                            String m1="";
                                            m1=Genere_Msg_Debug();
                                            Envoi_SMS_Debug(m1,phone);
                                 }


                                 // msg de seuil decrement
                                 if ((NULL != strstr(message,MESSAGE_DMD_SEUIL_DEC
))) {
                                            //Serial.println("Msg seuil dec" );
                                            Seuil--;
                                            String m1="";
                                            m1=Genere_Msg_Debug();
                                            Envoi_SMS_Debug(m1,phone);
                                 }

                                 // msg de demande de re init total
                                 if ((NULL !=
strstr(message,MESSAGE_DMD_REBOOT ))) {
                                     //Serial.println("Msg reboot" );
                                     Envoi_SMS_Debug(MSG_REBOOT,phone);
                                     delay(5000);
                                     resetFunc();                    // relance du
programme
                                  }
                        } // le correspondant est ok

                } // fin boucle cherche correspondant

               } // fin boucle recherche SMS
}     // fin lecture SMS
```

```
/* Lire le flash de la cloture  */
void Lire_Flash(){
    ValeurFlash=analogRead(lightPin); // lit la valeur du flash
     if ( ValeurFlash > FLASH )  {      // si on voit le flash incrémente le
compteur
               FlashVu++;
     }
}


// genere un entete de SMS standard à tous les msgs

String Entete(){

  String me="";
  gprs.getDateTime(DateTime);           // recup la date depuis sim900
  me="Sup Cloture ";
  me+=VERSION;
  me+="\n";
  me+=DateTime;
  me+="\n";
  me+="Depuis: ";
  me+=Synchro_Temps();
  me+="\n";
  me+="Site: ";
  me+=LIEU;
  me+="\n";
  return me;
}

// revoi le temps écoulé depuis le lancement
String  Synchro_Temps(){
    //Serial.println("Synchro_Temps");
    char  mt[24];
    gprs.getDateTime(DateTime);           // recup la date depuis sim900
    myDate.tm_year = (((int) DateTime[0] -48 ) *10 ) + ( (int) DateTime[1]-48);
    myDate.tm_mon=(((int) DateTime[3] -48 ) *10 ) + ( (int) DateTime[4]-48)-1;
    myDate.tm_mday=(((int) DateTime[6] -48 ) *10 ) + ( (int) DateTime[7]-48);
    myDate.tm_hour=(((int) DateTime[9] -48 ) *10 ) + ( (int) DateTime[10]-48);
    myDate.tm_min=(((int) DateTime[12] -48 ) *10 ) + ( (int) DateTime[13]-48);
    myDate.tm_sec=(((int) DateTime[15] -48 ) *10 ) + ( (int) DateTime[16]-48);
    // conversion en timestamp
    timestamp_courant = mktime( & myDate );
    // calcul du timestamp depuis le lancement
    timestamp_ecoule=timestamp_courant-timestamp_debut;
    //Serial.println(timestamp_ecoule);
    // met au format lisible
    myDate= *localtime(&timestamp_ecoule);
    sprintf(mt,"%02dj %02d:%02d:%02d",myDate.tm_mday-
1,myDate.tm_hour,myDate.tm_min,myDate.tm_sec);
    //Serial.println(mt);
    return mt;
}

/* calcul et stocke l'heure de lancement */
void Init_Heure_Debut(){
    //Serial.println("Init_Heure_Debut");
    myDate.tm_year = (((int) DateTime[0] -48 ) *10 ) + ( (int) DateTime[1]-48);
    myDate.tm_mon=(((int) DateTime[3] -48 ) *10 ) + ( (int) DateTime[4]-48) -1 ;
    myDate.tm_mday=(((int) DateTime[6] -48 ) *10 ) + ( (int) DateTime[7]-48);
    myDate.tm_hour=(((int) DateTime[9] -48 ) *10 ) + ( (int) DateTime[10]-48);
    myDate.tm_min=(((int) DateTime[12] -48 ) *10 ) + ( (int) DateTime[13]-48);
    myDate.tm_sec=(((int) DateTime[15] -48 ) *10 ) + ( (int) DateTime[16]-48);
    timestamp_debut = mktime( & myDate );
```

```
}

// cacul des heures pour envoi de sms d'infos systématique
bool Test_Horaire_Envoi_SMS(){
      gprs.getDateTime(DateTime);           // recup la date depuis sim900
      myDate.tm_hour=(((int) DateTime[9] -48 ) *10 ) + ( (int) DateTime[10]-48);
      myDate.tm_min=(((int) DateTime[12] -48 ) *10 ) + ( (int) DateTime[13]-48);
      myDate.tm_sec=(((int) DateTime[15] -48 ) *10 ) + ( (int) DateTime[16]-48);
      // compare l'heure courante et les valeurs prévues
      if ( ((myDate.tm_hour == HEURE_ENVOI_SMS_1) || (myDate.tm_hour ==
HEURE_ENVOI_SMS_2) ) &&
          ((myDate.tm_min < 2) && (myDate.tm_sec < 15 )) ) {
                return true;
            }
      return false;
}
```