```
// IMPORTANT: Adafruit_TFTLCD LIBRARY MUST BE SPECIFICALLY
// CONFIGURED FOR EITHER THE TFT SHIELD OR THE BREAKOUT BOARD.
// SEE RELEVANT COMMENTS IN Adafruit_TFTLCD.h FOR SETUP.
////Technical support:goodtft@163.com

// Smartpoker V1.0
//06-12-2018
// Slatkin : slatkin@hotmail.com
// Merci aux contributeurs pour leurs bibliotheques
// gestion de l'ecran :  http://osoyoo.com/2016/08/03/2-4tftlcd-for-arduino-uno-and-mega2560/
// librairie : http://osoyoo.com/wp-content/uploads/2016/08/Install-libraries.rar
// exemples :

// gestion des lecteurs RFID Weigand
//librairie : https://github.com/monkeyboard/Wiegand-Protocol-Library-for-Arduino/blob/master/
Wiegand.h
// Les ports utilisées sont 18 et 19 sur l'arduino MEGA - Obligatoire


//le module Wifi ESP01-S

// voir le site smartpoker.jimbo.com

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>
#include <TouchScreen.h>
#include <Wiegand.h>
#include <SoftwareSerial.h>

const String SMARTPOKER_SERIE=" SMP001";
const String SMARTPOKER_TYPE="SMP";
const String SMARTPOKER_VERSION=" 1.0";
const String SMARTPOKER_DATE="06/12/2018";

// The control pins for the LCD can be assigned to any digital or
// analog pins...but we'll use the analog pins as this allows us to
// double up the pins with the touch screen (see the TFT paint example).
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0

#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin

// When using the BREAKOUT BOARD only, use these 8 data lines to the LCD:
// For the Arduino Uno, Duemilanove, Diecimila, etc.:
//   D0 connects to digital pin 8  (Notice these are
//   D1 connects to digital pin 9   NOT in order!)
//   D2 connects to digital pin 2
//   D3 connects to digital pin 3
//   D4 connects to digital pin 4
```

```
//   D5 connects to digital pin 5
//   D6 connects to digital pin 6
//   D7 connects to digital pin 7
// For the Arduino Mega, use digital pins 22 through 29
// (on the 2-row header at the end of the board).

// Assign human-readable names to some common 16-bit color values:
#define  BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF

// Color definitions
#define ILI9341_BLACK       0x0000      /*   0,   0,   0 */
#define ILI9341_NAVY        0x000F      /*   0,   0, 128 */
#define ILI9341_DARKGREEN   0x03E0      /*   0, 128,   0 */
#define ILI9341_DARKCYAN    0x03EF      /*   0, 128, 128 */
#define ILI9341_MAROON      0x7800      /* 128,   0,   0 */
#define ILI9341_PURPLE      0x780F      /* 128,   0, 128 */
#define ILI9341_OLIVE       0x7BE0      /* 128, 128,   0 */
#define ILI9341_LIGHTGREY   0xC618      /* 192, 192, 192 */
#define ILI9341_DARKGREY    0x7BEF      /* 128, 128, 128 */
#define ILI9341_BLUE        0x001F      /*   0,   0, 255 */
#define ILI9341_GREEN       0x07E0      /*   0, 255,   0 */
#define ILI9341_CYAN        0x07FF      /*   0, 255, 255 */
#define ILI9341_RED         0xF800      /* 255,   0,   0 */
#define ILI9341_MAGENTA     0xF81F      /* 255,   0, 255 */
#define ILI9341_YELLOW      0xFFE0      /* 255, 255,   0 */
#define ILI9341_WHITE       0xFFFF      /* 255, 255, 255 */
#define ILI9341_ORANGE      0xFD20      /* 255, 165,   0 */
#define ILI9341_GREENYELLOW 0xAFE5      /* 173, 255,  47 */
#define ILI9341_PINK        0xF81F
/******************* UI details */
#define BUTTON_X 40
#define BUTTON_Y 100
#define BUTTON_W 60
#define BUTTON_H 30
#define BUTTON_SPACING_X 20
#define BUTTON_SPACING_Y 20
#define BUTTON_TEXTSIZE 2

// text box where numbers go
#define TEXT_X 10
#define TEXT_Y 10
#define TEXT_W 220
#define TEXT_H 50
#define TEXT_TSIZE 3
#define TEXT_TCOLOR ILI9341_MAGENTA
```

```
// the data (phone #) we store in the textfield
#define TEXT_LEN 12
char textfield[TEXT_LEN+1] = "";
uint8_t textfield_i=0;
String Stringfield="";
String Stringcmd="";

#define YP A3  // must be an analog pin, use "An" notation!
#define XM A2  // must be an analog pin, use "An" notation!
#define YM 9   // can be a digital pin
#define XP 8   // can be a digital pin

#define TS_MINX 150
#define TS_MINY 120
#define TS_MAXX 920
#define TS_MAXY 940
// We have a status line for like, is FONA working
#define STATUS_X 10
#define STATUS_Y 65
#define MINPRESSURE 10
#define MAXPRESSURE 1000


Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
// If using the shield, all control and data lines are fixed, and
// a simpler declaration can optionally be used:
// Adafruit_TFTLCD tft;

Adafruit_GFX_Button buttons[15];

/* create 15 buttons, in classic candybar phone style */
char buttonlabels[15][5] = {"Send", "Alin", "Out", "Fold","Chck","Call","x2  ", "x2.5","x3
","1/4p","1/2p","1/3p","3/4p","Pot ","->"};
char buttoncommand[15]
[5]={"0","52","0","55","117","62","106","107","109","112","113","114","115","116","0"};
char buttonlabels2[15][5] = {"Send", "Clr", "Out", "1", "2", "3", "4", "5", "6", "7", "8", "9", "Niv",
"0", "->" };

uint16_t buttoncolors[15] = {ILI9341_DARKGREEN, ILI9341_DARKGREY, ILI9341_RED,
                             ILI9341_BLUE, ILI9341_BLUE, ILI9341_BLUE,
                             ILI9341_BLUE, ILI9341_BLUE, ILI9341_BLUE,
                             ILI9341_BLUE, ILI9341_BLUE, ILI9341_BLUE,
                            ILI9341_BLUE, ILI9341_BLUE, ILI9341_ORANGE};
 unsigned long start = micros();
// TextSize
//=6 -> 6 caracteres
//=5 -> 8 caracteres
//=4 -> 10 caracteres
//=3 -> 13 caracteres
//=2 -> 20 caracteres
//=1 -> 40 caracteres
```

```
int change_page=0;
int change_niveau=0;
int page=-1;

/***ESP*****/
// Communication serie avec l'ESP
// Sur la MEGA utilisez uniquement pour RX:
// 10, 11, 12, 13, 50, 51, 52, 53, 62, 63, 64, 65, 66, 67, 68, 69
SoftwareSerial ESP01(52, 53); // RX, TX
 const byte numChars = 32;
 char receivedChars[numChars];
 boolean newData = false;


/****RFID ***/
WIEGAND wg;
long unsigned int lecteur1=0;
long unsigned int lecteur2=0;
long unsigned int Tag=0;
int pos=1;
int trouve=1;

/***Gestion des echanges avec l'arduino **/
String cmdtoarduino="";
String cmdfromarduino="";
String inputstring="";
char inChar;
String inputstringRX="";
char inCharRX;
bool stringComplete=false;
bool stringCompleteRX=false;
unsigned long cpt=0;
unsigned long maxcpt=1000000;
bool initwifi=true;
bool iprecu=false;

//* watchtdog
bool initwatchdog=false;
bool watchdogrecu=false;
unsigned long cptwatchdog=0;
unsigned long cptmaxwatchdog=1000000;
int retrywatchdog=0;
int retrymaxwatchdog=3;

/************************** Setup ************************/
void setup(void) {
  Serial.begin(9600);
  Serial2.begin(9600); // init RX2,TX2 pour l'ESP01
  clavier0a();
  Serial.println(F("TFT LCD test"));
```

```
#ifdef USE_ADAFRUIT_SHIELD_PINOUT
  Serial.println(F("Using Adafruit 2.4\" TFT Arduino Shield Pinout"));
#else
  Serial.println(F("Using Adafruit 2.4\" TFT Breakout Board Pinout"));
#endif

  Serial.print("TFT size is "); Serial.print(tft.width()); Serial.print("x"); Serial.println(tft.height());

  tft.reset();
  uint16_t identifier = tft.readID();
  if(identifier==0x0101)
      identifier=0x9341;

/* A conserver pour debug si besoin
    if(identifier == 0x9325) {
    Serial.println(F("Found ILI9325 LCD driver"));
  } else if(identifier == 0x4535) {
    Serial.println(F("Found LGDP4535 LCD driver"));
  }else if(identifier == 0x9328) {
    Serial.println(F("Found ILI9328 LCD driver"));
  } else if(identifier == 0x7575) {
    Serial.println(F("Found HX8347G LCD driver"));
  } else if(identifier == 0x9341) {
    Serial.println(F("Found ILI9341 LCD driver"));
  } else if(identifier == 0x8357) {
    Serial.println(F("Found HX8357D LCD driver"));
  } else {
    Serial.print(F("Unknown LCD driver chip: "));
    Serial.println(identifier, HEX);
    //Serial.println(F("If using the Adafruit 2.4\" TFT Arduino shield, the line:"));
    //Serial.println(F("  #define USE_ADAFRUIT_SHIELD_PINOUT"));
    //Serial.println(F("should appear in the library header (Adafruit_TFT.h)."));
    //Serial.println(F("If using the breakout board, it should NOT be #defined!"));
    //Serial.println(F("Also if using the breakout, double-check that all wiring"));
    //Serial.println(F("matches the tutorial."));
    return;
  }
*/

  tft.begin(identifier);
  Serial.println("Affichage Ecran0a");
  clavier0a();
  Serial.println("init lecteur RFID");
  wg.begin(18,19);
  delay(2000);
  clavier0b();
} //end setup

void loop(void) {

if (initwifi==true)
   {
```

```
    Serial.println(" debut initialisation Wifi2");
    page=0;
    Serial2.println('c');
    initwifi=false;
  }

if (iprecu==true)
   {
    Serial.println(" Envoi 1er Watchdog");
    page=0;
    Serial2.println('w');
    iprecu=false;
    initwatchdog=false;
  }

//Watchdog
cptwatchdog++;
if (cptwatchdog >= cptmaxwatchdog)
   {
    if (watchdogrecu==true)
       {
        cptwatchdog=0;
        watchdogrecu=false;
        retrywatchdog=0;
        Serial2.println("w"); //reemission d'un watchdog
       }
    else
      {retrywatchdog++;
      // ecrire un message : watchdog KO
      }
   }// end if if (cptwatchdog >= cptmaxwatchdog)

 if (retrywatchdog>=retrymaxwatchdog)
   {
      // ecrirer un message dans status //pas de connexion avec le serveur
   } //endif(retrywatchdog>=3)


// lecture des ports series

while (Serial2.available()) //les données sont recus de l'ESP01
    {
      // get the new byte:
     inChar = (char)Serial2.read();
   // add it to the inputString:
      inputstring += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
      if (inChar == '\n')
      {
       stringComplete = true;
       int nbcar=inputstring.length()-1;
```

```
       String temp=inputstring.substring(0,nbcar);
       Serial.println(temp);
      if (page==0)
      {
       if (inputstring.substring(0,3)=="Mac")
           {
            Serial.println("MAC recu");
            tft.println("MAC");
            tft.println(temp);
           }
       if (inputstring.substring(0,2)=="IP")
          {
           Serial.println("IP recu");
           tft.println("IP");
           tft.println(temp);
           iprecu=true;
           initwatchdog=true;
          }
      if (inputstring.substring(0,11)=="Watchdog OK")
          {
           Serial.println("Watchdog recu");
           tft.println("Connexion avec serveur");
           tft.println("OK");
           watchdogrecu=true;
           page=1;
           delay(5000);
           clavier1(); //changement de clavier
           status("Bienvenu");
          }
      }
       inputstring="";
      }//end if Inchar
     } //endwhile Serial2.available

while (Serial.available()) // les données sont envoyées à l'ESP01
    {
      // get the new byte:
     inCharRX = (char)Serial.read();
   // add it to the inputString:
      inputstringRX += inCharRX;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
      if (inCharRX == '\n')
      {
       stringCompleteRX = true;
       Serial2.println(inputstringRX);
       inputstringRX="";
      }//end if Inchar
    } //endwhile Serial.available

//lecture RFID
  if(wg.available())
```

```
 {
    //je dois reemettre les tag lu pour gerer le mode on/off des cartes de controles
  Serial.print("Wiegand HEX = ");
  Serial.print(wg.getCode(),HEX);
  Serial.print(", DECIMAL = ");
  Serial.print(wg.getCode());
  Serial.print(", Type W");
  Serial.println(wg.getWiegandType());
  Tag=wg.getCode();
  cmdfromarduino="Tag=";
  cmdfromarduino+=Tag;
  if (pos==1)
       {
        if (Tag==lecteur1)
          {
           pos=1;
          }
        else if (Tag==lecteur2)
           {
            pos=2;
           }
        else
           {
            lecteur1=Tag; pos=2;
           }
       } //ind if pos==1
  else
      {
        if (Tag==lecteur1)
          {
           pos=1;
          }
        else if (Tag==lecteur2)
           {
            pos=2;
           }
        else
           {
            lecteur2=Tag; pos=1;
           }
      }
  cmdfromarduino+="&Posid=";
  cmdfromarduino+=pos;
  Serial.print("cmdfromarduino :");
  Serial.println(cmdfromarduino);
  Serial2.println(cmdfromarduino);


} //if(wg.available())

// Gestion du changement de page
  if (change_page==1)
```

```
    {
      if(page==1)
       {
        page=2;
        clavier2();
       }
       else
       {
        page=1;
        clavier1();
       }
       change_page=0;
    }



//clavier
 /*TSPoint p;
  p = ts.getPoint();
  */

  digitalWrite(13, HIGH);
  TSPoint p = ts.getPoint();
  digitalWrite(13, LOW);

  // if sharing pins, you'll need to fix the directions of the touchscreen pins
  //pinMode(XP, OUTPUT);
  pinMode(XM, OUTPUT);
  pinMode(YP, OUTPUT);
  //pinMode(YM, OUTPUT);

  // we have some minimum pressure we consider 'valid'
  // pressure of 0 means no pressing!

 // p = ts.getPoint();
  /*
  if (ts.bufferSize()) {

  } else {
    // this is our way of tracking touch 'release'!
    p.x = p.y = p.z = -1;
  }*/

  // Scale from ~0->4000 to tft.width using the calibration #'s
  /*
  if (p.z != -1) {
    p.x = map(p.x, TS_MINX, TS_MAXX, 0, tft.width());
    p.y = map(p.y, TS_MINY, TS_MAXY, 0, tft.height());
    Serial.print("("); Serial.print(p.x); Serial.print(", ");
    Serial.print(p.y); Serial.print(", ");
    Serial.print(p.z); Serial.println(") ");
  }*/
```

```
 if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
  // scale from 0->1023 to tft.width
  p.x = map(p.x, TS_MINX, TS_MAXX, tft.width(), 0);
  p.y = map(p.y, TS_MINY, TS_MAXY, tft.height(), 0);
 }

// go thru all the buttons, checking if they were pressed
for (uint8_t b=0; b<15; b++) {
  if (buttons[b].contains(p.x, p.y)) {
    //Serial.print("Pressing: "); Serial.println(b);
    buttons[b].press(true);  // tell the button it is pressed
  } else {
    buttons[b].press(false);  // tell the button it is NOT pressed
  }
}

// now we can ask the buttons if their state has changed
for (uint8_t b=0; b<15; b++) {
  if (buttons[b].justReleased()) {
    // Serial.print("Released: "); Serial.println(b);
    buttons[b].drawButton();  // draw normal
  }

  if (buttons[b].justPressed()) {
      buttons[b].drawButton(true);  // draw invert!

      // cas page 1**********************
  if (page==1)
   {
      // if a numberpad button, append the relevant # to the textfield
      // clr button
       if (b == 0) {
         status(F("Envoi"));
        cmdfromarduino="Tag=";
        cmdfromarduino+=Stringcmd;
        Serial.print("cmdfromarduino :");
        Serial.println(cmdfromarduino);
        Serial2.println(cmdfromarduino);
        Stringfield="";
        //efface ecran
      }

      if (b == 1) {
         status(F("Clear"));
      }
      if (b == 2) {
        Stringfield= "Out";
      }
      if (b >= 3)
        {
        Stringfield = buttonlabels[b];
        Stringcmd =  buttoncommand[b];
```

```
      Serial.println("Stringfield");
      Serial.println(Stringfield);
      } //if (b >= 3)

    //changement de page
    if (b == 14)
      {
        change_page=1;
      }

    // update the current text field
    Serial.println(Stringfield);
    tft.setCursor(TEXT_X + 2, TEXT_Y+10);
    tft.print("       ");
    delay(100);
    tft.setCursor(TEXT_X + 2, TEXT_Y+10);
    tft.setTextColor(TEXT_TCOLOR, ILI9341_BLACK);
    tft.setTextSize(TEXT_TSIZE);
    tft.print(Stringfield);
    Stringfield="";
  } //if page=1

  /**********************cas page 2 **********************/
if (change_page==2)
  {
    if (b == 0) {
      status(F("Envoi"));
      cmdfromarduino="Tag=";
      cmdfromarduino+=Stringfield;
      Serial.print("cmdfromarduino :");
      Serial.println(cmdfromarduino);
      Serial2.println(cmdfromarduino);
    }
    // clr button! delete char
    if (b == 1) {

      textfield[textfield_i] = 0;
      if (textfield > 0) {
        textfield_i--;
        textfield[textfield_i] = ' ';
      }
    }

    // its always OK to just hang up
    if (b == 2) {
      status(F("Out"));

    }

   // if a numberpad button, append the relevant # to the textfield
    if (b >= 3) {
```

```
          //saisie Niveau
          if (b == 12) {
            change_niveau=1;
            goto suite;
          }

          //changement de page
          if (b == 14) {
            change_page=1;
            goto suite ;
          }

          if (textfield_i < TEXT_LEN)
          {
            textfield[textfield_i] = buttonlabels2[b][0]; //
            textfield_i++;
            textfield[textfield_i] = 0; // zero terminate
          }
        } //if (b >= 3)
    suite:

        // update the current text field
        Serial.println(textfield);
        tft.setCursor(TEXT_X + 2, TEXT_Y+10);
        tft.setTextColor(TEXT_TCOLOR, ILI9341_BLACK);
        tft.setTextSize(TEXT_TSIZE);
        tft.print(textfield);
    }//page==2

      delay(100); // UI debouncing
    } //if (buttons[b].justPressed())
  } // for (uint8_t b=0; b<15; b++)


} //end loop


/*************** Fonction ecran  **************/
int clavier0a()
{

 tft.setRotation(1);
 tft.fillScreen(BLACK);
 tft.setTextColor(WHITE);
 tft.setTextSize(2);
 tft.setCursor(1,2);
 tft.println("SMARTPOKER_SERIE:");
 tft.println(SMARTPOKER_SERIE);
 tft.println("SMARTPOKER_TYPE:");
 tft.println(SMARTPOKER_TYPE);
 tft.println("SMARTPOKER_VERSION:");
 tft.println(SMARTPOKER_VERSION);
```

```
 tft.println("SMARTPOKER_DATE:");
 tft.println(SMARTPOKER_DATE);
 tft.println("");
 tft.setTextSize(2);
 tft.println("Visitez le site");
 tft.setTextSize(3);
 tft.println("");
 tft.println("smartpoker.fr");
 delay(5000);
}

int clavier0b()
{
 Serial.println("Affichage clavier0b");
 tft.setRotation(1);
 tft.fillScreen(BLACK);
 tft.setTextColor(WHITE);
 tft.setTextSize(2);
 tft.setCursor(1,2);
 delay(4000);
}

int clavier1()
{
  tft.setRotation(0);
  tft.fillScreen(BLACK);


  // create buttons
  for (uint8_t row=0; row<5; row++) {
    for (uint8_t col=0; col<3; col++) {
      buttons[col + row*3].initButton(&tft,
BUTTON_X+col*(BUTTON_W+BUTTON_SPACING_X),
                 BUTTON_Y+row*(BUTTON_H+BUTTON_SPACING_Y),    // x, y, w, h, outline, fill,
text
                  BUTTON_W, BUTTON_H, ILI9341_WHITE, buttoncolors[col+row*3],
ILI9341_WHITE,
                  buttonlabels[col + row*3],1);
      buttons[col + row*3].drawButton();
    }
  }

  // create 'text field'
  tft.drawRect(TEXT_X, TEXT_Y, TEXT_W, TEXT_H, ILI9341_WHITE);
}

int clavier2()
{
  tft.setRotation(0);
  tft.fillScreen(BLACK);

  // create buttons
```

```
  for (uint8_t row=0; row<5; row++) {
    for (uint8_t col=0; col<3; col++) {
      buttons[col + row*3].initButton(&tft,
BUTTON_X+col*(BUTTON_W+BUTTON_SPACING_X),
                 BUTTON_Y+row*(BUTTON_H+BUTTON_SPACING_Y),    // x, y, w, h, outline, fill,
text
                  BUTTON_W, BUTTON_H, ILI9341_WHITE, buttoncolors[col+row*3],
ILI9341_WHITE,
                  buttonlabels2[col + row*3],1);
      buttons[col + row*3].drawButton();
    }
  }

  // create 'text field'
  tft.drawRect(TEXT_X, TEXT_Y, TEXT_W, TEXT_H, ILI9341_WHITE);
}


// Print something in the mini status bar with either flashstring
void status(const __FlashStringHelper *msg) {
  tft.fillRect(STATUS_X, STATUS_Y, 240, 8, ILI9341_BLACK);
  tft.setCursor(STATUS_X, STATUS_Y);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(1);
  tft.print(msg);
}
// or charstring
void status(char *msg) {
  tft.fillRect(STATUS_X, STATUS_Y, 240, 8, ILI9341_BLACK);
  tft.setCursor(STATUS_X, STATUS_Y);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(1);
  tft.print(msg);
}

//*************************************************
//************************** Fonction port serie
//*************************************************

 void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;
  while (Serial.available() > 0 && newData == false) {
   rc = Serial.read();
   if (recvInProgress == true) {
    if (rc != endMarker) {
     receivedChars[ndx] = rc;
     ndx++;
     if (ndx >= numChars) {
```

```
      ndx = numChars - 1;
     }
    }
    else {
     receivedChars[ndx] = '\0'; // terminate the string
     recvInProgress = false;
     ndx = 0;
     newData = true;
    }
   }
   else if (rc == startMarker) {
    recvInProgress = true;
   }
  }
 } //end recvWithStartEndMarkers() {


 void showNewData() {
  if (newData == true) {
   Serial.print("Commande recue : ");
   String sCmd = String(receivedChars);
   sCmd.toLowerCase();
   sCmd.trim();
   Serial.println(sCmd);
   int ind1 = sCmd.indexOf('=');
   String sComd = sCmd.substring(0, ind1);
   String sVal = sCmd.substring(ind1 + 1);
   Serial.println("Commande : " + sComd + " ; Valeur : " + sVal);
   if (sComd.equals("led"))
   {
     if (sVal.equals("on"))
      {
     //digitalWrite(ledPin, HIGH);
     // Action1
    //else if (sVal.equals("off"))
     //digitalWrite(ledPin, LOW);
   // Action2
      } //(sVal.equals("on"))
   }  //(sComd.equals("led"))

   newData = false;
  }
 } //end  showNewData()

int decode_cmd(String cmdfromarduino)
{
  int nbcar=0;
  bool controle_format=false;
  bool debug=true;

  /*structure : cmd=xx?yyyyyy
     tag=  :mot clé de longeur fixe , 4 caracteres
```

```
    xxx    : est un commande, on peut la laisser au format string, mais 3 caracteres uniquements
    cela correspond aux codes de la table Tcontroles
    &data=    : mot clé , c'est un separateur sur un caractere
    yyyyy : données variables liées à la commande
*/
  if (debug)
  {
     Serial.print ("cmdfromarduino: ");
     Serial.println(cmdfromarduino);
  }
 /*controle du format de la trame*/
  nbcar = cmdfromarduino.length()-1; //j'enleve le caractere /n du compteur
  if (debug)
  {
     Serial.print ("Nb car : ");
     Serial.println(nbcar);
  }
  if (nbcar <=8) //if faut rajouter le caractere /n dans
    {
     Serial.println("Nombre de caracteres insuffisant");
     controle_format=false;
    }
 /*recherche cmd=, en premiere position */
  if (cmdfromarduino.startsWith("Tag=",1))
  {
     Serial.println("mot clé tag OK");
     controle_format=true;
  }

  Serial.println("cmd=01"); //ecran 01
  Serial.println("cmd=05"); //efface ecran
        Serial.println("cmd=06"); //textesize=1;
        Serial.print("cmd=10?"); //Status
        //Serial.println(payload.substring(debut,payload.length()));


} //end decode_cmd
```