ESP32 CAM SD

Recuperer Fichiers sur esp32 cam programme 001

esp32camSD.ino

```
/*********************************************************************

L'ESP32-CAM présente une page web qui permet de prendre des photos
et de les enregistrer sur une carte SD, de visionner les photos
déjà présentes sur la carte, et de supprimer les photos non-désirées.

Pour plus d'informations:

http://electroniqueamateur.blogspot.com/2020/07/esp32-cam-gestion-distance-de-la-carte.html

*********************************************************************/

#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include "FS.h"       // manipulation de fichiers
#include "SD_MMC.h"   // carte SD
#include "esp_camera.h" // caméra!

// écrivez le nom et le mot de passe de votre réseau WIFI
const char* ssid = "**********";
const char* password = "**********";

WebServer server(80);

static bool cartePresente = false;

int numero_fichier = 0; // numéro de la photo (nom du fichier)


// prise de la photo et création du fichier jpeg

void enregistrer_photo (void)
{
  char adresse[20] = ""; // chemin d'accès du fichier .jpeg
  camera_fb_t * fb = NULL; // frame buffer

  // prise de la photo
  fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Echec de la prise de photo.");
```

```
    return;
  }

  numero_fichier = numero_fichier + 1;

  // enregitrement du fichier sur la carte SD

  sprintf (adresse, "/%d.jpg", numero_fichier);

  fs::FS &fs = SD_MMC;
  File file = fs.open(adresse, FILE_WRITE);

  if (!file) {
    Serial.println("Echec lors de la creation du fichier.");
  }
  else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Fichier enregistre: %s\n", adresse);
  }
  file.close();
  esp_camera_fb_return(fb);

   // on affiche un message de confirmation
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);

  server.send(200,  "text/html", "");

  WiFiClient client = server.client();

  server.sendContent("<h1>Une nouvelle photo a &eacute;t&eacute;
prise.</h1>");
  server.sendContent("<p><a href = / > Retour &agrave; la liste des
fichiers </a></p>");

}

void returnOK() {
  server.send(200, "text/plain", "");
}

void returnFail(String msg) {
  server.send(500, "text/plain", msg + "\r\n");
}

// Affichage d'un fichier présent sur la carte
bool loadFromSdCard(String path) {
  String dataType = "text/plain";

  if (path == "/") {
```

```
    printDirectory();
  }
  else {

    if (path.endsWith(".src")) {
      path = path.substring(0, path.lastIndexOf("."));
    } else if (path.endsWith(".htm")) {
      dataType = "text/html";
    } else if (path.endsWith(".css")) {
      dataType = "text/css";
    } else if (path.endsWith(".js")) {
      dataType = "application/javascript";
    } else if (path.endsWith(".png")) {
      dataType = "image/png";
    } else if (path.endsWith(".gif")) {
      dataType = "image/gif";
    } else if (path.endsWith(".jpg")) {
      dataType = "image/jpeg";
    } else if (path.endsWith(".ico")) {
      dataType = "image/x-icon";
    } else if (path.endsWith(".xml")) {
      dataType = "text/xml";
    } else if (path.endsWith(".pdf")) {
      dataType = "application/pdf";
    } else if (path.endsWith(".zip")) {
      dataType = "application/zip";
    }

    fs::FS &fs = SD_MMC;

    File dataFile = fs.open(path.c_str());

    if (!dataFile) {
      return false;
    }

    if (server.hasArg("download")) {
      dataType = "application/octet-stream";
    }

    if (server.streamFile(dataFile, dataType) != dataFile.size()) {
      Serial.println("Sent less data than expected!");
    }

    dataFile.close();

  }
  return true;
}
```

```
// utilisé lors de la suppression d'un fichier
void deleteRecursive(String path) {

  fs::FS &fs = SD_MMC;
  File file = fs.open((char *)path.c_str());
  if (!file.isDirectory()) {
    file.close();
    fs.remove((char *)path.c_str());
    return;
  }

  file.rewindDirectory();
  while (true) {
    File entry = file.openNextFile();
    if (!entry) {
      break;
    }
    String entryPath = path + "/" + entry.name();
    if (entry.isDirectory()) {
      entry.close();
      deleteRecursive(entryPath);
    } else {
      entry.close();
      fs.remove((char *)entryPath.c_str());
    }
    yield();
  }

  fs.rmdir((char *)path.c_str());
  file.close();
}

// suppression d'un fichier

void handleDelete() {

  fs::FS &fs = SD_MMC;

  if (server.args() == 0) {
    return returnFail("Mauvais arguments?");
  }
  String path = server.arg(0);
  if (path == "/" || !fs.exists((char *)path.c_str())) {
    returnFail("BAD PATH");
    return;
  }
  deleteRecursive(path);

  // on affiche un message de confirmation
```

```
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);

  server.send(200,  "text/html", "");

  WiFiClient client = server.client();

  server.sendContent("<h1>Le fichier a &eacute;t&eacute;
supprim&eacute;</h1>");
  server.sendContent("<p><a href = / > Retour &agrave; la liste des
fichiers </a></p>");

}

// Affichage du contenu de la carte
void printDirectory() {

  fs::FS &fs = SD_MMC;
  String path = "/";

  File dir = fs.open((char *)path.c_str());
  path = String();
  if (!dir.isDirectory()) {
    dir.close();
    return returnFail("PAS UN REPERTOIRE");
  }
  dir.rewindDirectory();
  server.setContentLength(CONTENT_LENGTH_UNKNOWN);

  server.send(200,  "text/html", "");

  WiFiClient client = server.client();

  server.sendContent("<h1>Prise de photo</h1>");

  server.sendContent("<br> <form action='/clic' method='GET'> <INPUT
type='submit' value='Prendre une photo'></form><br> ");

  server.sendContent("<h1>Contenu de la carte SD</h1>");

  for (int cnt = 0; true; ++cnt) {
    File entry = dir.openNextFile();
    if (!entry) {
      break;
    }

    String output;

    output += "<a href = ";
    output += entry.name();
    output += "> ";
    output += entry.name();
```

```
    // on ajoute un bouton delete:
    output += "</a>          <a href =
/delete?url=";
    output += entry.name();
    output += "> [Supprimer] </a> <br>";

    server.sendContent(output);
    entry.close();
  }
  dir.close();
}

// on tente d'afficher le fichier demandé. Sinon, message d'erreur
void handleNotFound() {
  if (cartePresente && loadFromSdCard(server.uri())) {
    return;
  }
  String message = "Carte SD non detectee ou action imprevue\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " NAME:" + server.argName(i) + "\n VALUE:" +
server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
  Serial.print(message);
}

void setup(void) {

  // définition des broches de la caméra pour le modèle AI Thinker -
ESP32-CAM
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = 5;
  config.pin_d1 = 18;
  config.pin_d2 = 19;
  config.pin_d3 = 21;
  config.pin_d4 = 36;
  config.pin_d5 = 39;
  config.pin_d6 = 34;
  config.pin_d7 = 35;
```

```
  config.pin_xclk = 0;
  config.pin_pclk = 22;
  config.pin_vsync = 25;
  config.pin_href = 23;
  config.pin_sscb_sda = 26;
  config.pin_sscb_scl = 27;
  config.pin_pwdn = 32;
  config.pin_reset = -1;

  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;  //YUV422|GRAYSCALE|RGB565|JPEG
  config.frame_size = FRAMESIZE_VGA;  //
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
  config.jpeg_quality = 10;  // 0-63 ; plus bas = meilleure qualité
  config.fb_count = 2; // nombre de frame buffers

  // initialisation de la caméra
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Echec de l'initialisation de la camera, erreur
0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();

  // connexion au WIFI

  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connexion au reseau Wifi ");
  Serial.println(ssid);

  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED && i++ < 20) {//wait 10 seconds
    delay(500);
  }
  if (i == 21) {
    Serial.print("Impossible de se connecter au reseau ");
    Serial.println(ssid);
    while (1) {
      delay(500);
    }
  }
  Serial.print("Connecte a l'adresse IP: ");
  Serial.println(WiFi.localIP());

  // initialisation du web server
  server.on("/delete", HTTP_GET, handleDelete);
  server.on("/clic", HTTP_GET, enregistrer_photo);
```

```
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("Serveur HTTP en fonction.");

  // initialisation de la carte micro SD
  if (SD_MMC.begin()) {
    uint8_t cardType = SD_MMC.cardType();
    if (cardType != CARD_NONE) {
      Serial.println("Carte SD Initialisee.");
      cartePresente = true;
    }
  }
}

void loop(void) {
  server.handleClient();
}
```

Esp32 cam Serveur

Page projet ESP32Cam Serveur EN

Esp32CamServeur001.ino

```
/*********
  Rui Santos
  Complete project details at
https://RandomNerdTutorials.com/esp32-cam-video-streaming-web-server-ca
mera-home-assistant/

  IMPORTANT!!!
   - Select Board "AI Thinker ESP32-CAM"
   - GPIO 0 must be connected to GND to upload a sketch
   - After connecting GPIO 0 to GND, press the ESP32-CAM on-board RESET
button to put your board in flashing mode

  Permission is hereby granted, free of charge, to any person obtaining
a copy
  of this software and associated documentation files.

  The above copyright notice and this permission notice shall be
included in all
  copies or substantial portions of the Software.
*********/

#include "esp_camera.h"
#include <WiFi.h>
```

```
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_cntl_reg.h"  //disable brownout problems
#include "esp_http_server.h"

//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM
Model and M5STACK WITHOUT PSRAM
#define CAMERA_MODEL_AI_THINKER
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM

// Not tested with this model
//#define CAMERA_MODEL_WROVER_KIT

#if defined(CAMERA_MODEL_WROVER_KIT)
  #define PWDN_GPIO_NUM    -1
  #define RESET_GPIO_NUM   -1
  #define XCLK_GPIO_NUM    21
  #define SIOD_GPIO_NUM    26
  #define SIOC_GPIO_NUM    27

  #define Y9_GPIO_NUM      35
  #define Y8_GPIO_NUM      34
  #define Y7_GPIO_NUM      39
  #define Y6_GPIO_NUM      36
  #define Y5_GPIO_NUM      19
  #define Y4_GPIO_NUM      18
  #define Y3_GPIO_NUM       5
  #define Y2_GPIO_NUM       4
  #define VSYNC_GPIO_NUM   25
  #define HREF_GPIO_NUM    23
  #define PCLK_GPIO_NUM    22

#elif defined(CAMERA_MODEL_M5STACK_PSRAM)
  #define PWDN_GPIO_NUM     -1
  #define RESET_GPIO_NUM    15
  #define XCLK_GPIO_NUM     27
  #define SIOD_GPIO_NUM     25
  #define SIOC_GPIO_NUM     23

  #define Y9_GPIO_NUM       19
  #define Y8_GPIO_NUM       36
```

```
  #define Y7_GPIO_NUM       18
  #define Y6_GPIO_NUM       39
  #define Y5_GPIO_NUM        5
  #define Y4_GPIO_NUM       34
  #define Y3_GPIO_NUM       35
  #define Y2_GPIO_NUM       32
  #define VSYNC_GPIO_NUM    22
  #define HREF_GPIO_NUM     26
  #define PCLK_GPIO_NUM     21

#elif defined(CAMERA_MODEL_M5STACK_WITHOUT_PSRAM)
  #define PWDN_GPIO_NUM     -1
  #define RESET_GPIO_NUM    15
  #define XCLK_GPIO_NUM     27
  #define SIOD_GPIO_NUM     25
  #define SIOC_GPIO_NUM     23

  #define Y9_GPIO_NUM       19
  #define Y8_GPIO_NUM       36
  #define Y7_GPIO_NUM       18
  #define Y6_GPIO_NUM       39
  #define Y5_GPIO_NUM        5
  #define Y4_GPIO_NUM       34
  #define Y3_GPIO_NUM       35
  #define Y2_GPIO_NUM       17
  #define VSYNC_GPIO_NUM    22
  #define HREF_GPIO_NUM     26
  #define PCLK_GPIO_NUM     21

#elif defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1
  #define XCLK_GPIO_NUM      0
  #define SIOD_GPIO_NUM     26
  #define SIOC_GPIO_NUM     27

  #define Y9_GPIO_NUM       35
  #define Y8_GPIO_NUM       34
  #define Y7_GPIO_NUM       39
  #define Y6_GPIO_NUM       36
  #define Y5_GPIO_NUM       21
  #define Y4_GPIO_NUM       19
  #define Y3_GPIO_NUM       18
  #define Y2_GPIO_NUM        5
  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22
#else
  #error "Camera model not selected"
```

```
#endif

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-
replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-
Length: %u\r\n\r\n";

httpd_handle_t stream_httpd = NULL;

static esp_err_t stream_handler(httpd_req_t *req){
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if(res != ESP_OK){
    return res;
  }

  while(true){
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      res = ESP_FAIL;
    } else {
      if(fb->width > 400){
        if(fb->format != PIXFORMAT_JPEG){
          bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf,
&_jpg_buf_len);
          esp_camera_fb_return(fb);
          fb = NULL;
          if(!jpeg_converted){
            Serial.println("JPEG compression failed");
            res = ESP_FAIL;
          }
        } else {
          _jpg_buf_len = fb->len;
          _jpg_buf = fb->buf;
        }
      }
    }
    if(res == ESP_OK){
      size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART,
_jpg_buf_len);
      res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
    if(res == ESP_OK){
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf,
```

```
_jpg_buf_len);
    }
    if(res == ESP_OK){
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
    }
    if(fb){
      esp_camera_fb_return(fb);
      fb = NULL;
      _jpg_buf = NULL;
    } else if(_jpg_buf){
      free(_jpg_buf);
      _jpg_buf = NULL;
    }
    if(res != ESP_OK){
      break;
    }
    //Serial.printf("MJPG: %uB\n",(uint32_t)(_jpg_buf_len));
  }
  return res;
}

void startCameraServer(){
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;

  httpd_uri_t index_uri = {
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
  };

  //Serial.printf("Starting web server on port: '%d'\n",
config.server_port);
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &index_uri);
  }
}

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout
detector

  Serial.begin(115200);
  Serial.setDebugOutput(false);

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
```

```
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // Camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  // Wi-Fi connection
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.print("Camera Stream Ready! Go to: http://");
  Serial.print(WiFi.localIP());

  // Start streaming web server
  startCameraServer();
}
```

```
void loop() {
  delay(1);
}
```

ESP32 CAm et Photos ---> sur Google Drive

ESP 32 Cam Photos sur google drive

espcam-Gdrive

Doc FR ESP32 Cam Google drive

esp32camPhotosDrive001.ino

```
/*
   L'ESP32-CAM prend une photo et l'enregistre sur Google Drive.

   Basé sur le sketch de Guillermo Sampallo:
   https://github.com/gsampallo/esp32cam-gdrive

   Plus d'informations:

https://electroniqueamateur.blogspot.com/2020/05/enregistrement-sur-goo
gle-drive-des.html

*/

#include "esp_camera.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>

const char* ssid     = "**********";   // nom du réseau Wifi
const char* password = "**********";   // mot de passe du réseau Wifi
// à remplacer par l'url de votre Google Apps Script:
String urlScript = "/macros/s/******************/exec";

const int delaiImage = 60000; // nombre de millisecondes entre 2
enregistrements succesifs
const int delaiReponse = 30000; // nombre de millisecondes max
d'attente pour la réponse de Google

const char* host = "script.google.com";
String nomFichier = "filename=ESP32-CAM.jpg";
String mimeType = "&mimetype=image/jpeg";
String monImage = "&data=";
```

```
void setup()
{
  Serial.begin(115200);
  delay(10);

  WiFi.mode(WIFI_STA);

  Serial.println("");
  Serial.print("Connexion a ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("Adresse IP: ");
  Serial.println(WiFi.localIP());

  Serial.println("");

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = 5;
  config.pin_d1 = 18;
  config.pin_d2 = 19;
  config.pin_d3 = 21;
  config.pin_d4 = 36;
  config.pin_d5 = 39;
  config.pin_d6 = 34;
  config.pin_d7 = 35;
  config.pin_xclk = 0;
  config.pin_pclk = 22;
  config.pin_vsync = 25;
  config.pin_href = 23;
  config.pin_sscb_sda = 26;
  config.pin_sscb_scl = 27;
  config.pin_pwdn = 32;
  config.pin_reset = -1;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;  //
UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA
  config.jpeg_quality = 10;
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
```

```
    Serial.printf("Echec de l'initialisation 0x%x", err);
    delay(1000);
    ESP.restart();
  }
}

void loop() {
  enregistreImage();
  delay(delaiImage);
}

void enregistreImage() {
  Serial.println("Connexion a " + String(host));

  WiFiClientSecure client;

  if (client.connect(host, 443)) {
    Serial.println("Connection reussie");

    camera_fb_t * fb = NULL;
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Echec de la prise de photo ");
      delay(1000);
      ESP.restart();
      return;
    }

    char *input = (char *)fb->buf;
    char output[base64_enc_len(3)];
    String imageFile = "";
    for (int i = 0; i < fb->len; i++) {
      base64_encode(output, (input++), 3);
      if (i % 3 == 0) imageFile += urlencode(String(output));
    }
    String Data = nomFichier + mimeType + monImage;

    esp_camera_fb_return(fb);

    Serial.println("Image envoyee a Google Drive.");

    client.println("POST " + urlScript + " HTTP/1.1");
    client.println("Host: " + String(host));
    client.println("Content-Length: " + String(Data.length() +
imageFile.length()));
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.println();
    client.print(Data);
    int Index;
```

```
    for (Index = 0; Index < imageFile.length(); Index = Index + 1000) {
      client.print(imageFile.substring(Index, Index + 1000));
    }

    Serial.println("Attente de la reponse.");
    long int StartTime = millis();
    while (!client.available()) {
      Serial.print(".");
      delay(100);
      if ((StartTime + waitingTime) < millis()) {
        Serial.println();
        Serial.println("Pas de reponse.");
        break;
      }
    }
    Serial.println();
    while (client.available()) {
      Serial.print(char(client.read()));
    }
  } else {
    Serial.println("Echec de la connexion a " + String(host) + ".");
  }
  client.stop();
}

/************** Encodage de l'url ******************************/

// urlencode sert à remplacer les caractères indésirables
// dans une url (par exemple, remplacer un espace par %20`)

//https://github.com/zenmanenergy/ESP8266-Arduino-Examples/
String urlencode(String str)
{
  String encodedString = "";
  char c;
  char code0;
  char code1;
  char code2;
  for (int i = 0; i < str.length(); i++) {
    c = str.charAt(i);
    if (c == ' ') {
      encodedString += '+';
    } else if (isalnum(c)) {
      encodedString += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) {
        code1 = (c & 0xf) - 10 + 'A';
      }
      c = (c >> 4) & 0xf;
      code0 = c + '0';
```

```
      if (c > 9) {
        code0 = c - 10 + 'A';
      }
      code2 = '\0';
      encodedString += '%';
      encodedString += code0;
      encodedString += code1;
    }
    yield();
  }
  return encodedString;
}


// *************** Encodage de l'image en base64
*************************
//Copyright (c) 2013 Adam Rudd.
//https://github.com/adamvr/arduino-base64

const char PROGMEM b64_alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                    "abcdefghijklmnopqrstuvwxyz"
                                    "0123456789+/";

int base64_encode(char *output, char *input, int inputLen) {
  int i = 0, j = 0;
  int encLen = 0;
  unsigned char a3[3];
  unsigned char a4[4];

  while (inputLen--) {
    a3[i++] = *(input++);
    if (i == 3) {
      a3_to_a4(a4, a3);

      for (i = 0; i < 4; i++) {
        output[encLen++] = pgm_read_byte(&b64_alphabet[a4[i]]);
      }
      i = 0;
    }
  }

  if (i) {
    for (j = i; j < 3; j++) {
      a3[j] = '\0';
    }

    a3_to_a4(a4, a3);

    for (j = 0; j < i + 1; j++) {
```

```
      output[encLen++] = pgm_read_byte(&b64_alphabet[a4[j]]);
    }

    while ((i++ < 3)) {
      output[encLen++] = '=';
    }
  }
  output[encLen] = '\0';
  return encLen;
}

int base64_enc_len(int plainLen) {
  int n = plainLen;
  return (n + 2 - ((n + 2) % 3)) / 3 * 4;
}

inline void a3_to_a4(unsigned char * a4, unsigned char * a3) {
  a4[0] = (a3[0] & 0xfc) >> 2;
  a4[1] = ((a3[0] & 0x03) << 4) + ((a3[1] & 0xf0) >> 4);
  a4[2] = ((a3[1] & 0x0f) << 2) + ((a3[2] & 0xc0) >> 6);
  a4[3] = (a3[2] & 0x3f);
}
```