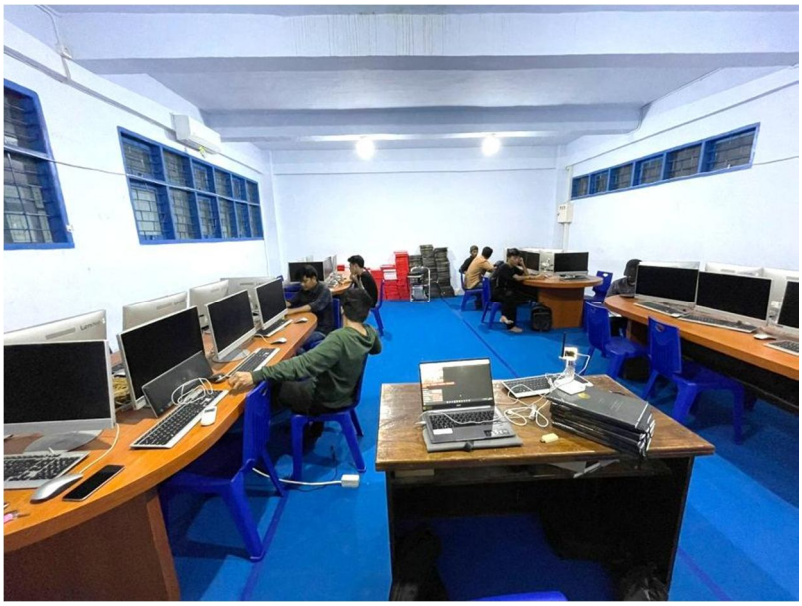
Lampiran 1. Dokumentasi





Lampiran 2. SCRIPT ARDUINO

```
#include "esp_log.h"
#include "esp_http_server.h"
#include "esp_camera.h"
#include "sensor.h"

const char* ssid = "server";          //Wi-Fi
ssid
const char* password = "123456789";
//Wi-Fi password

const char* apssid = "Recorder";
const char* appassword = "12345678";

String devstr =  "Rekaman_";
boolean recordOnce = false;
boolean resetfilegroup = false;

int avi_length = 180;
int framesize = FRAMESIZE_CIF;        //
UXGA(1600x1200)|SXGA(1280x1024)|XG
A(1024x768)|SVGA(800x600)|VGA(640x4
80)|CIF(400x296)|QVGA(320x240)|HQVG
A(240x176)|QQVGA(160x120)
int quality = 10;                     //  10 ~ 63

String Feedback="",recordMessage="";
String
Command="",cmd="",P1="",P2="",P3="",P
4="",P5="",P6="",P7="",P8="",P9="";
byte
ReceiveState=0,cmdState=1,strState=1,quest
ionstate=0,equalstate=0,semicolonstate=0;

static const char vernum[] = "v50lpmod";
char devname[30];
#define Lots_of_Stats 1

int framesizeconfig = FRAMESIZE_UXGA;
// default framesize
int qualityconfig = 5;                // default
quality
int buffersconfig = 3;                // default
buffers
int frame_interval = 0;               // record at
full speed
int speed_up_factor = 1;              // play at
realtime
int stream_delay = 500;               //
minimum of 500 ms delay between frames
int MagicNumber = 12;                 // change
this number to reset the eprom in your esp32
for file numbers

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~
bool configfile = false;
bool InternetOff = true;
bool reboot_now = false;
String cssid;
String cpass;
String czone;

TaskHandle_t the_camera_loop_task;
TaskHandle_t the_sd_loop_task;
TaskHandle_t the_streaming_loop_task;

SemaphoreHandle_t wait_for_sd;
SemaphoreHandle_t sd_go;

long current_frame_time;
long last_frame_time;

#define fbs 8 // was 64 -- how many kb of
static ram for psram -> sram buffer for sd
write
uint8_t framebuffer_static[fbs * 1024 + 20];

typedef struct {
      size_t size; //number of values used for
filtering
      size_t index; //current value index
      size_t count; //value count
      int sum;
      int * values; //array to be filled with
values
} ra_filter_t;

typedef struct {
      httpd_req_t *req;
      size_t len;
} jpg_chunking_t;

#define PART_BOUNDARY
"123456789000000000000987654321"
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";

static ra_filter_t ra_filter;
httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

static ra_filter_t * ra_filter_init(ra_filter_t *
filter, size_t sample_size){
   memset(filter, 0, sizeof(ra_filter_t));

   filter->values = (int *)malloc(sample_size
* sizeof(int));
   if(!filter->values){
      return NULL;
   }
   memset(filter->values, 0, sample_size *
sizeof(int));

   filter->size = sample_size;
   return filter;
}

static int ra_filter_run(ra_filter_t * filter, int
value){
   if(!filter->values){
      return value;
   }
   filter->sum -= filter->values[filter-
>index];
   filter->values[filter->index] = value;
   filter->sum += filter->values[filter-
>index];
   filter->index++;
   filter->index = filter->index % filter-
>size;
   if (filter->count < filter->size) {
      filter->count++;
   }
   return filter->sum / filter->count;
}

// 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

camera_fb_t * fb_curr = NULL;
camera_fb_t * fb_next = NULL;

#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "nvs.h"
#include "soc/soc.h"
#include "soc/cpu.h"
#include "soc/rtc_cntl_reg.h"

static esp_err_t cam_err;
float most_recent_fps = 0;
int most_recent_avg_framesize = 0;

uint8_t* framebuffer;
int framebuffer_len;

int first = 1;
long frame_start = 0;
long frame_end = 0;
long frame_total = 0;
long frame_average = 0;
long loop_average = 0;
long loop_total = 0;
long total_frame_data = 0;
long last_frame_length = 0;
int done = 0;
long avi_start_time = 0;
long avi_end_time = 0;
int start_record = 0;

int we_are_already_stopped = 0;
long total_delay = 0;
long bytes_before_last_100_frames = 0;
long time_before_last_100_frames = 0;

long time_in_loop = 0;
long time_in_camera = 0;
long time_in_sd = 0;
long time_in_good = 0;
long time_total = 0;
long time_in_web1 = 0;
long time_in_web2 = 0;
long delay_wait_for_sd = 0;
long wait_for_cam = 0;
```

```
int do_it_now = 0;

// MicroSD
#include "driver/sdmmc_host.h"
#include "driver/sdmmc_defs.h"
#include "sdmmc_cmd.h"
#include "esp_vfs_fat.h"
#include "FS.h"
#include <SD_MMC.h>

File logfile;
File avifile;
File idxfile;

char avi_file_name[100];

static int i = 0;
uint16_t frame_cnt = 0;
uint16_t remnant = 0;
uint32_t length = 0;
uint32_t startms;
uint32_t elapsedms;
uint32_t uVideoLen = 0;

int bad_jpg = 0;
int extend_jpg = 0;
int normal_jpg = 0;

int file_number = 0;
int file_group = 0;
long boot_time = 0;

long totalp;
long totalw;

#define BUFFSIZE 512

uint8_t buf[BUFFSIZE];

#define AVIOFFSET 240 // AVI main
header length

unsigned long movi_size = 0;
unsigned long jpeg_size = 0;
unsigned long idx_offset = 0;

uint8_t zero_buf[4] = {0x00, 0x00, 0x00,
0x00};
uint8_t dc_buf[4] = {0x30, 0x30, 0x64,
0x63};    // "00dc"
uint8_t dc_and_zero_buf[8] = {0x30, 0x30,
0x64, 0x63, 0x00, 0x00, 0x00, 0x00};

uint8_t avi1_buf[4] = {0x41, 0x56, 0x49,
0x31};    // "AVI1"

uint8_t idx1_buf[4] = {0x69, 0x64, 0x78,
0x31};    // "idx1"

struct frameSizeStruct {
  uint8_t frameWidth[2];
  uint8_t frameHeight[2];
};

void the_camera_loop (void* pvParameter);
void the_sd_loop (void* pvParameter);
void delete_old_stuff();

static const frameSizeStruct frameSizeData[]
= {
  {{0x60, 0x00}, {0x60, 0x00}}, //
FRAMESIZE_96X96,    // 96x96
  {{0xA0, 0x00}, {0x78, 0x00}}, //
FRAMESIZE_QQVGA,    // 160x120
  {{0xB0, 0x00}, {0x90, 0x00}}, //
FRAMESIZE_QCIF,     // 176x144
  {{0xF0, 0x00}, {0xB0, 0x00}}, //
FRAMESIZE_HQVGA,    // 240x176
  {{0xF0, 0x00}, {0xF0, 0x00}}, //
FRAMESIZE_240X240,  // 240x240
  {{0x40, 0x01}, {0xF0, 0x00}}, //
FRAMESIZE_QVGA,     // 320x240
framessize
  {{0x90, 0x01}, {0x28, 0x01}}, //
FRAMESIZE_CIF,      // 400x296       bytes
per buffer required in psram - quality must
be higher number (lower quality) than config
quality
  {{0xE0, 0x01}, {0x40, 0x01}}, //
FRAMESIZE_HVGA,     // 480x320       low
qual  med qual  high quality
  {{0x80, 0x02}, {0xE0, 0x01}}, //
FRAMESIZE_VGA,      // 640x480   8   11+
##  6-10  ##  0-5         indoor(56,COUNT=3)
(56,COUNT=2)          (56,count=1)
                                        //
38,400    61,440    153,600
  {{0x20, 0x03}, {0x58, 0x02}}, //
FRAMESIZE_SVGA,     // 800x600   9
  {{0x00, 0x04}, {0x00, 0x03}}, //
FRAMESIZE_XGA,      // 1024x768  10
  {{0x00, 0x05}, {0xD0, 0x02}}, //
FRAMESIZE_HD,       // 1280x720  11
115,200   184,320   460,800     (11)50.000
25.4fps   (11)50.000 12fps    (11)50,000
12.7fps
  {{0x00, 0x05}, {0x00, 0x04}}, //
FRAMESIZE_SXGA,     // 1280x1024 12
  {{0x40, 0x06}, {0xB0, 0x04}}, //
FRAMESIZE_UXGA,     // 1600x1200 13
240,000   384,000   960,000
```

```
  // 3MP Sensors
  {{0x80, 0x07}, {0x38, 0x04}}, //
FRAMESIZE_FHD,      // 1920x1080 14
259,200   414,720   1,036,800   (11)210,000
5.91fps
  {{0xD0, 0x02}, {0x00, 0x05}}, //
FRAMESIZE_P_HD,     //  720x1280 15
  {{0x60, 0x03}, {0x00, 0x06}}, //
FRAMESIZE_P_3MP,    //  864x1536 16
  {{0x00, 0x08}, {0x00, 0x06}}, //
FRAMESIZE_QXGA,     // 2048x1536 17
393,216   629,146   1,572,864
  // 5MP Sensors
  {{0x00, 0x0A}, {0xA0, 0x05}}, //
FRAMESIZE_QHD,      // 2560x1440 18
460,800   737,280   1,843,200   (11)400,000
3.5fps    (11)330,000 1.95fps
  {{0x00, 0x0A}, {0x40, 0x06}}, //
FRAMESIZE_WQXGA,    // 2560x1600 19
  {{0x38, 0x04}, {0x80, 0x07}}, //
FRAMESIZE_P_FHD,    // 1080x1920 20
  {{0x00, 0x0A}, {0x80, 0x07}}  //
FRAMESIZE_QSXGA,    // 2560x1920 21
614,400   983,040   2,457,600   (15)425,000
3.25fps   (15)382,000 1.7fps  (15)385,000
1.7fps

};

const int avi_header[AVIOFFSET]
PROGMEM = {
  0x52, 0x49, 0x46, 0x46, 0xD8, 0x01,
0x0E, 0x00, 0x41, 0x56, 0x49, 0x20, 0x4C,
0x49, 0x53, 0x54,
  0xD0, 0x00, 0x00, 0x00, 0x68, 0x64, 0x72,
0x6C, 0x61, 0x76, 0x69, 0x68, 0x38, 0x00,
0x00, 0x00,
  0xA0, 0x86, 0x01, 0x00, 0x80, 0x66, 0x01,
0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
0x00, 0x00,
  0x64, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x80, 0x02, 0x00, 0x00, 0xe0, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x4C, 0x49, 0x53, 0x54, 0x84, 0x00,
0x00, 0x00,
  0x73, 0x74, 0x72, 0x6C, 0x73, 0x74, 0x72,
0x68, 0x30, 0x00, 0x00, 0x00, 0x76, 0x69,
0x64, 0x73,
  0x4D, 0x4A, 0x50, 0x47, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x0A, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x73, 0x74,
0x72, 0x66,
  0x28, 0x00, 0x00, 0x00, 0x28, 0x00, 0x00,
0x00, 0x80, 0x02, 0x00, 0x00, 0xe0, 0x01,
0x00, 0x00,
  0x01, 0x00, 0x18, 0x00, 0x4D, 0x4A,
0x50, 0x47, 0x00, 0x84, 0x03, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x49, 0x4E,
0x46, 0x4F,
  0x10, 0x00, 0x00, 0x00, 0x6A, 0x61,
0x6D, 0x65, 0x73, 0x7A, 0x61, 0x68, 0x61,
0x72, 0x79, 0x20,
  0x76, 0x35, 0x30, 0x20, 0x4C, 0x49, 0x53,
0x54, 0x00, 0x01, 0x0E, 0x00, 0x6D, 0x6F,
0x76, 0x69,
};

static void inline print_quartet(unsigned long
i, File fd) {

  uint8_t y[4];
  y[0] = i % 0x100;
  y[1] = (i >> 8) % 0x100;
  y[2] = (i >> 16) % 0x100;
  y[3] = (i >> 24) % 0x100;
  size_t i1_err = fd.write(y , 4);
}

static void inline print_2quartet(unsigned
long i, unsigned long j, File fd) {

  uint8_t y[8];
  y[0] = i % 0x100;
  y[1] = (i >> 8) % 0x100;
  y[2] = (i >> 16) % 0x100;
  y[3] = (i >> 24) % 0x100;
  y[4] = j % 0x100;
  y[5] = (j >> 8) % 0x100;
  y[6] = (j >> 16) % 0x100;
  y[7] = (j >> 24) % 0x100;
  size_t i1_err = fd.write(y , 8);
}

void major_fail() {

  Serial.println(" ");
  logfile.close();

  for  (int i = 0;  i < 3; i++) {
```

```
    for (int j = 0; j < 3; j++) {
      digitalWrite(33, LOW);   delay(150);
      digitalWrite(33, HIGH);  delay(150);
    }
    delay(1000);

    for (int j = 0; j < 3; j++) {
      digitalWrite(33, LOW);  delay(500);
      digitalWrite(33, HIGH); delay(500);
    }
    delay(1000);
    Serial.print("Major Fail  "); Serial.print(i);
Serial.print(" / "); Serial.println(3);
    recordMessage = "Major Fail "+String(i);
  }

  ESP.restart();
}

static esp_err_t config_camera() {

  camera_config_t config;

  //Serial.println("config camera");

  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;     //
10000000 or 20000000 -- 100 is faster with
v1.04  // 200 is faster with v1.06 // 16500000
is an option

  config.pixel_format =
PIXFORMAT_JPEG;

  Serial.printf("Frame config %d, quality
config %d, buffers config %d\n",
framesizeconfig, qualityconfig,
buffersconfig);
  config.frame_size =
(framesize_t)framesizeconfig;
  config.jpeg_quality = qualityconfig;
  config.fb_count = buffersconfig;

  esp_err_t cam_err = ESP_FAIL;
  int attempt = 5;
  while (attempt && cam_err != ESP_OK) {
    cam_err = esp_camera_init(&config);
    if (cam_err != ESP_OK) {
      Serial.printf("Camera init failed with
error 0x%x", cam_err);
      digitalWrite(PWDN_GPIO_NUM, 1);
      delay(500);
      digitalWrite(PWDN_GPIO_NUM, 0); //
power cycle the camera (OV2640)
      attempt--;
    }
  }

  if (Lots_of_Stats) {
    Serial.printf("After  camera config ...");
    Serial.printf("Internal Total heap %d,
internal Free Heap %d, ",
ESP.getHeapSize(), ESP.getFreeHeap());
    Serial.printf("SPIRam Total heap   %d,
SPIRam Free Heap   %d\n",
ESP.getPsramSize(), ESP.getFreePsram());
  }

  if (cam_err != ESP_OK) {
    major_fail();
  }

  sensor_t * ss = esp_camera_sensor_get();

  ///ss->set_hmirror(ss, 1);        // 0 = disable
, 1 = enable
  ///ss->set_vflip(ss, 1);          // 0 = disable ,
1 = enable

  Serial.printf("\nCamera started correctly,
Type is %x (hex) of 9650, 7725, 2640, 3660,
5640\n\n", ss->id.PID);

  ss->set_hmirror(ss, 0);        // 0 = disable , 1
= enable
  ss->set_quality(ss, quality);
  ss->set_framesize(ss,
(framesize_t)framesize);

  ss->set_brightness(ss, 1);  //up the
blightness just a bit
```

```
  ss->set_saturation(ss, -2); //lower the
saturation

  delay(800);
  for (int j = 0; j < 4; j++) {
    camera_fb_t * fb = esp_camera_fb_get();
// get_good_jpeg();
    if (!fb) {
      Serial.println("Camera Capture Failed");
    } else {
      //Serial.print("Pic, len="); Serial.print(fb-
>len);
      //Serial.printf(", new fb %X\n", (long)fb-
>buf);
      esp_camera_fb_return(fb);
      delay(50);
    }
  }
}

static esp_err_t init_sdcard()
{

  int succ = SD_MMC.begin("/sdcard", true);
  if (succ) {
    Serial.printf("SD_MMC Begin: %d\n",
succ);
    uint8_t cardType =
SD_MMC.cardType();
    Serial.print("SD_MMC Card Type: ");
    if (cardType == CARD_MMC) {
      Serial.println("MMC");
    } else if (cardType == CARD_SD) {
      Serial.println("SDSC");
    } else if (cardType == CARD_SDHC) {
      Serial.println("SDHC");
    } else {
      Serial.println("UNKNOWN");
    }

    uint64_t cardSize = SD_MMC.cardSize()
/ (1024 * 1024);
    Serial.printf("SD_MMC Card Size:
%lluMB\n", cardSize);
    Serial.println("");
  } else {
    Serial.printf("Failed to mount SD card
VFAT filesystem. \n");
    Serial.println("Do you have an SD Card
installed?");
    Serial.println("Check pin 12 and 13, not
grounded, or grounded with 10k
resistors!\n\n");
    major_fail();
  }
  return ESP_OK;
}

void read_config_file() {

  File config_file =
SD_MMC.open("/config.txt", "r");
  if (!config_file) {
    Serial.println("Failed to open config_file
for reading");
  } else {
    String junk;
    Serial.println("Reading config.txt");
    String cname =
config_file.readStringUntil(' ');
    junk = config_file.readStringUntil('\n');
    int cframesize = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cquality = config_file.parseInt();
    junk = config_file.readStringUntil('\n');

    int cframesizeconfig =
config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cqualityconfig = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cbuffersconfig = config_file.parseInt();
    junk = config_file.readStringUntil('\n');

    int clength = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cinterval = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cspeedup = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cstreamdelay = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    int cinternet = config_file.parseInt();
    junk = config_file.readStringUntil('\n');
    String czone =
config_file.readStringUntil(' ');
    junk = config_file.readStringUntil('\n');
    cssid = config_file.readStringUntil(' ');
    junk = config_file.readStringUntil('\n');
    cpass = config_file.readStringUntil(' ');
    junk = config_file.readStringUntil('\n');
    config_file.close();

    Serial.printf("=========  Data fram
config.txt  =========\n");
    Serial.printf("Name %s\n", cname);
logfile.printf("Name %s\n", cname);
```

```
    Serial.printf("Framesize %d\n",
cframesize); logfile.printf("Framesize
%d\n", cframesize);
    Serial.printf("Quality %d\n", cquality);
logfile.printf("Quality %d\n", cquality);
    Serial.printf("Framesize config %d\n",
cframesizeconfig); logfile.printf("Framesize
config%d\n", cframesizeconfig);
    Serial.printf("Quality config %d\n",
cqualityconfig); logfile.printf("Quality
config%d\n", cqualityconfig);
    Serial.printf("Buffers config %d\n",
cbuffersconfig); logfile.printf("Buffers
config %d\n", cbuffersconfig);
    Serial.printf("Length %d\n", clength);
logfile.printf("Length %d\n", clength);
    Serial.printf("Interval %d\n", cinterval);
logfile.printf("Interval %d\n", cinterval);
    Serial.printf("Speedup %d\n", cspeedup);
logfile.printf("Speedup %d\n", cspeedup);
    Serial.printf("Streamdelay %d\n",
cstreamdelay); logfile.printf("Streamdelay
%d\n", cstreamdelay);
    Serial.printf("Internet %d\n", cinternet);
logfile.printf("Internet %d\n", cinternet);
    //Serial.printf("Zone len %d, %s\n",
czone.length(), czone); //logfile.printf("Zone
len %d, %s\n", czone.length(), czone);
    Serial.printf("Zone len %d\n",
czone.length()); logfile.printf("Zone len
%d\n", czone.length());
    Serial.printf("ssid %s\n", cssid);
logfile.printf("ssid %s\n", cssid);
    Serial.printf("pass %s\n", cpass);
logfile.printf("pass %s\n", cpass);


    framesize = cframesize;
    quality = cquality;
    framesizeconfig = cframesizeconfig;
    qualityconfig = cqualityconfig;
    buffersconfig = cbuffersconfig;
    avi_length = clength;
    frame_interval = cinterval;
    speed_up_factor = cspeedup;
    stream_delay = cstreamdelay;
    configfile = true;

    cname.toCharArray(devname,
cname.length() + 1);
  }
}

void listDir( const char * dirname, uint8_t
levels) {
  Serial.printf("Listing directory: %s\n", "/");

  File root = SD_MMC.open("/");
  if (!root) {
    Serial.println("Failed to open directory");
    return;
  }
  if (!root.isDirectory()) {
    Serial.println("Not a directory");
    return;
  }

  File filex = root.openNextFile();
  while (filex) {
    if (filex.isDirectory()) {
      Serial.print("  DIR : ");
      Serial.println(filex.name());
      if (levels) {
        listDir( filex.name(), levels - 1);
      }
    } else {
      Serial.print("  FILE: ");
      Serial.print(filex.name());
      Serial.print("  SIZE: ");
      Serial.println(filex.size());
    }
    filex = root.openNextFile();
  }
}

void delete_old_stuff() {

  Serial.printf("Total space: %lluMB\n",
SD_MMC.totalBytes() / (1024 * 1024));
  Serial.printf("Used space: %lluMB\n",
SD_MMC.usedBytes() / (1024 * 1024));

  //listDir( "/", 0);

  float full = 1.0 * SD_MMC.usedBytes() /
SD_MMC.totalBytes();;
  if (full  <  0.8) {
    Serial.printf("DATA TIDAK ADA,
%.1f%% memory full\n", 100.0 * full);
  } else {
    Serial.printf("Memory %.1f%% penuh ...
hapus file lama\n", 100.0 * full);
    while (full > 0.8) {

      double del_number = 999999999;
      char del_numbername[50];

      File f = SD_MMC.open("/");
```

```
    File file = f.openNextFile();

    while (file) {
     //Serial.println(file.name());
     if (!file.isDirectory()) {

       char foldname[50];
       strcpy(foldname, file.name());
       for ( int x = 0; x < 50; x++) {
        if ( (foldname[x] >= 0x30 &&
foldname[x] <= 0x39) || foldname[x] ==
0x2E) {
         } else {
          if (foldname[x] != 0) foldname[x] =
0x20;
         }
       }

       double i = atof(foldname);
       if ( i > 0 && i < del_number) {
        strcpy (del_numbername,
file.name());
         del_number = i;
       }
      }
      file = f.openNextFile();

    }
    Serial.printf("lowest is Name is %s,
number is %f\n", del_numbername,
del_number);
    if (del_number < 999999999) {
     deleteFolderOrFile(del_numbername);
    }
    full = 1.0 * SD_MMC.usedBytes() /
SD_MMC.totalBytes();
    Serial.printf("Disk is %.1f%% full ... \n",
100.0 * full);
    f.close();
   }
  }
}

void deleteFolderOrFile(const char * val) {
 // Function provided by user @gemi254
 Serial.printf("Deleting : %s\n", val);
 File f = SD_MMC.open(val);
 if (!f) {
  Serial.printf("Failed to open %s\n", val);
  return;
 }

 if (f.isDirectory()) {
  File file = f.openNextFile();
  while (file) {
   if (file.isDirectory()) {
    Serial.print("  DIR : ");
    Serial.println(file.name());
   } else {
    Serial.print("  FILE: ");
    Serial.print(file.name());
    Serial.print("  SIZE: ");
    Serial.print(file.size());
    if (SD_MMC.remove(file.name())) {
     Serial.println(" deleted.");
    } else {
     Serial.println(" FAILED.");
    }
   }
   file = f.openNextFile();
  }
  f.close();
  //Remove the dir
  if (SD_MMC.rmdir(val)) {
   Serial.printf("Dir %s removed\n", val);
  } else {
   Serial.println("Remove dir failed");
  }

 } else {
  //Remove the file
  if (SD_MMC.remove(val)) {
   Serial.printf("File %s deleted\n", val);
  } else {
   Serial.println("Delete failed");
  }
 }
}

camera_fb_t *  get_good_jpeg() {

 camera_fb_t * fb;

 long start;
 int failures = 0;

 do {
  int fblen = 0;
  int foundffd9 = 0;
  long bp = millis();
  long mstart = micros();

  fb = esp_camera_fb_get();
  if (!fb) {
   Serial.println("Camera Capture Failed");
   failures++;
  } else {
   long mdelay = micros() - mstart;

   int get_fail = 0;
```

```
    totalp = totalp + millis() - bp;
    time_in_camera = totalp;

    fblen = fb->len;

    for (int j = 1; j <= 1025; j++) {
      if (fb->buf[fblen - j] != 0xD9) {
        // no d9, try next for
      } else {
        if (fb->buf[fblen - j - 1] == 0xFF ) {
          if (j == 1) {
            normal_jpg++;
          } else {
            extend_jpg++;
          }
          foundffd9 = 1;
          if (Lots_of_Stats) {
            if (j > 900) {
              Serial.print("Frame ");
Serial.print(frame_cnt); logfile.print("Frame
"); logfile.print(frame_cnt);
              Serial.print(", Len = ");
Serial.print(fblen); logfile.print(", Len = ");
logfile.print(fblen);
              //Serial.print(", Correct Len = ");
Serial.print(fblen - j + 1);
              Serial.print(", Extra Bytes = ");
Serial.println( j - 1); logfile.print(", Extra
Bytes = "); logfile.println( j - 1);
              logfile.flush();
            }

            if ( frame_cnt % 100 == 50) {
              Serial.printf("Frame %6d, len
%6d, extra  %4d, cam time %7d ",
frame_cnt, fblen, j - 1, mdelay / 1000);
              logfile.printf("Frame %6d, len
%6d, extra  %4d, cam time %7d ",
frame_cnt, fblen, j - 1, mdelay / 1000);
              do_it_now = 1;
            }
          }
          break;
        }
      }
    }

    if (!foundffd9) {
      bad_jpg++;
      Serial.printf("Bad jpeg, Frame %d, Len
= %d \n", frame_cnt, fblen);
      logfile.printf("Bad jpeg, Frame %d,
Len = %d\n", frame_cnt, fblen);
      esp_camera_fb_return(fb);
      failures++;

    } else {
      break;
    }
  }

  } while (failures < 10);

  if (failures == 10) {
    Serial.printf("10 failures");
    logfile.printf("10 failures");
    logfile.flush();

    sensor_t * ss = esp_camera_sensor_get();
    int qual = ss->status.quality ;
    ss->set_quality(ss, qual + 5);
    quality = qual + 5;
    Serial.printf("\n\nDecreasing quality due
to frame failures %d -> %d\n\n", qual, qual
+ 5);
    logfile.printf("\n\nDecreasing quality due
to frame failures %d -> %d\n\n", qual, qual
+ 5);
    delay(1000);

    start_record = 0;
    //reboot_now = true;
  }
  return fb;
}

#include <EEPROM.h>

struct eprom_data {
  int eprom_good;
  int file_group;
};

void do_eprom_read() {

  eprom_data ed;

  EEPROM.begin(200);
  EEPROM.get(0, ed);

  if (ed.eprom_good == MagicNumber) {
    Serial.println("Good settings in the
EPROM ");
    file_group = ed.file_group;
    file_group++;
    Serial.print("New File Group ");
Serial.println(file_group );
  } else {
```

```
    Serial.println("No settings in EPROM -
Starting with File Group 1 ");
    file_group = 1;
  }
  do_eprom_write();
  file_number = 1;
}

void do_eprom_write() {

  eprom_data ed;
  ed.eprom_good = MagicNumber;
  ed.file_group  = file_group;
  if (resetfilegroup==true) {
    file_group = 1;
    ed.file_group  = 1;   //reset file_group
    resetfilegroup=false;
  }

  Serial.println("Writing to EPROM, File
Group : " + String(file_group));
  Serial.println("");

  EEPROM.begin(200);
  EEPROM.put(0, ed);
  EEPROM.commit();
  EEPROM.end();
}


static esp_err_t start_avi() {

  long start = millis();

  Serial.println("Starting an avi ");

  sprintf(avi_file_name, "/%s%d.%03d.avi",
devname, file_group, file_number);

  file_number++;

  SD_MMC.end();
  SD_MMC.begin("/sdcard", true);
  avifile = SD_MMC.open(avi_file_name,
"w");
  idxfile = SD_MMC.open("/idx.tmp", "w");

  if (avifile) {
    Serial.printf("File open: %s\n",
avi_file_name);
    logfile.printf("File open: %s\n",
avi_file_name);
  }  else  {
    Serial.println("Could not open file");
    recordMessage = "Could not open file";
    major_fail();
  }

  if (idxfile)  {
    //Serial.printf("File open: %s\n",
"//idx.tmp");
  }  else  {
    Serial.println("DATA TIDAK DAPAT
DIBUKA /idx.tmp");
    major_fail();
  }

  for ( i = 0; i < AVIOFFSET; i++){
    char ch =
pgm_read_byte(&avi_header[i]);
    buf[i] = ch;
  }

  memcpy(buf + 0x40,
frameSizeData[framesize].frameWidth, 2);
  memcpy(buf + 0xA8,
frameSizeData[framesize].frameWidth, 2);
  memcpy(buf + 0x44,
frameSizeData[framesize].frameHeight, 2);
  memcpy(buf + 0xAC,
frameSizeData[framesize].frameHeight, 2);

  size_t err = avifile.write(buf,
AVIOFFSET);

  avifile.seek( AVIOFFSET, SeekSet);

  Serial.print(F("\nRecording "));
  Serial.print(avi_length);
  Serial.println(" seconds.");

  startms = millis();

  totalp = 0;
  totalw = 0;

  jpeg_size = 0;
  movi_size = 0;
  uVideoLen = 0;
  idx_offset = 4;

  bad_jpg = 0;
  extend_jpg = 0;
  normal_jpg = 0;

  time_in_loop = 0;
  time_in_camera = 0;
  time_in_sd = 0;
  time_in_good = 0;
  time_total = 0;
```

```
  time_in_web1 = 0;
  time_in_web2 = 0;
  delay_wait_for_sd = 0;
  wait_for_cam = 0;

  time_in_sd += (millis() - start);

  logfile.flush();
  avifile.flush();

}

static esp_err_t
another_save_avi(camera_fb_t * fb ) {

  long start = millis();

  int fblen;
  fblen = fb->len;

  int fb_block_length;
  uint8_t* fb_block_start;

  jpeg_size = fblen;

  remnant = (4 - (jpeg_size & 0x00000003))
& 0x00000003;

  long bw = millis();
  long frame_write_start = millis();

  framebuffer_static[0] = 0x30;       // "00dc"
  framebuffer_static[1] = 0x30;
  framebuffer_static[2] = 0x64;
  framebuffer_static[3] = 0x63;

  int jpeg_size_rem = jpeg_size + remnant;

  framebuffer_static[4] = jpeg_size_rem %
0x100;
  framebuffer_static[5] = (jpeg_size_rem >>
8) % 0x100;
  framebuffer_static[6] = (jpeg_size_rem >>
16) % 0x100;
  framebuffer_static[7] = (jpeg_size_rem >>
24) % 0x100;

  fb_block_start = fb->buf;

  if (fblen > fbs * 1024 - 8 ) {
    fb_block_length = fbs * 1024;
    fblen = fblen - (fbs * 1024 - 8);
    memcpy(framebuffer_static + 8,
fb_block_start, fb_block_length - 8);
    fb_block_start = fb_block_start +
fb_block_length - 8;

  } else {
    fb_block_length = fblen + 8  + remnant;
    memcpy(framebuffer_static + 8,
fb_block_start,  fblen);
    fblen = 0;
  }

  size_t err =
avifile.write(framebuffer_static,
fb_block_length);

  if (err != fb_block_length) {
    Serial.print("Error on avi write: err = ");
Serial.print(err);
    Serial.print(" len = ");
Serial.println(fb_block_length);
    logfile.print("Error on avi write: err = ");
logfile.print(err);
    logfile.print(" len = ");
logfile.println(fb_block_length);
  }

  while (fblen > 0) {

    if (fblen > fbs * 1024) {
      fb_block_length = fbs * 1024;
      fblen = fblen - fb_block_length;
    } else {
      fb_block_length = fblen  + remnant;
      fblen = 0;
    }

    memcpy(framebuffer_static,
fb_block_start, fb_block_length);

    size_t err =
avifile.write(framebuffer_static,
fb_block_length);

    if (err != fb_block_length) {
      Serial.print("Error on avi write: err = ");
Serial.print(err);
      Serial.print(" len = ");
Serial.println(fb_block_length);
    }

    fb_block_start = fb_block_start +
fb_block_length;
    delay(0);
  }

  movi_size += jpeg_size;
```

```
  uVideoLen += jpeg_size;
  long frame_write_end = millis();

  print_2quartet(idx_offset, jpeg_size,
idxfile);

  idx_offset = idx_offset + jpeg_size +
remnant + 8;

  movi_size = movi_size + remnant;

  if ( do_it_now == 1) {
    do_it_now = 0;
    Serial.printf(" sd time %4d -- \n",  millis()
- bw);
    logfile.printf(" sd time %4d -- \n",
millis() - bw);
    logfile.flush();
  }

  totalw = totalw + millis() - bw;
  time_in_sd += (millis() - start);

  avifile.flush();


} // end of another_pic_avi

static esp_err_t end_avi() {

  long start = millis();

  unsigned long current_end =
avifile.position();

  Serial.println("End of avi - closing the
files");
  logfile.println("End of avi - closing the
files");

  if (frame_cnt <  5 ) {
    Serial.println("Recording screwed up, less
than 5 frames, forget index\n");
    idxfile.close();
    avifile.close();
    int xx = remove("/idx.tmp");
    int yy = remove(avi_file_name);

  } else {

    elapsedms = millis() - startms;

    float fRealFPS = (1000.0f *
(float)frame_cnt) / ((float)elapsedms) *
speed_up_factor;

    float fmicroseconds_per_frame =
1000000.0f / fRealFPS;
    uint8_t iAttainedFPS = round(fRealFPS) ;
    uint32_t us_per_frame =
round(fmicroseconds_per_frame);


    avifile.seek( 4 , SeekSet);
    print_quartet(movi_size + 240 + 16 *
frame_cnt + 8 * frame_cnt, avifile);

    avifile.seek( 0x20 , SeekSet);
    print_quartet(us_per_frame, avifile);

    unsigned long max_bytes_per_sec = (1.0f
* movi_size * iAttainedFPS) / frame_cnt;

    avifile.seek( 0x24 , SeekSet);
    print_quartet(max_bytes_per_sec, avifile);

    avifile.seek( 0x30 , SeekSet);
    print_quartet(frame_cnt, avifile);

    avifile.seek( 0x8c , SeekSet);
    print_quartet(frame_cnt, avifile);

    avifile.seek( 0x84 , SeekSet);
    print_quartet((int)iAttainedFPS, avifile);

    avifile.seek( 0xe8 , SeekSet);
    print_quartet(movi_size + frame_cnt * 8 +
4, avifile);

    Serial.println(F("\n*** Video recorded
and saved ***\n"));

    Serial.printf("Recorded %5d frames in
%5d seconds\n", frame_cnt, elapsedms /
1000);
    Serial.printf("File size is %u bytes\n",
movi_size + 12 * frame_cnt + 4);
    Serial.printf("Adjusted FPS is %5.2f\n",
fRealFPS);
    Serial.printf("Max data rate is %lu
bytes/s\n", max_bytes_per_sec);
    Serial.printf("Frame duration is %d us\n",
us_per_frame);
    Serial.printf("Average frame length is %d
bytes\n", uVideoLen / frame_cnt);
    Serial.print("Average picture time (ms) ");
Serial.println( 1.0 * totalp / frame_cnt);
    Serial.print("Average write time (ms)   ");
Serial.println( 1.0 * totalw / frame_cnt );
```

```
    Serial.print("Normal jpg % ");
Serial.println( 100.0 * normal_jpg /
frame_cnt, 1 );
    Serial.print("Extend jpg % ");
Serial.println( 100.0 * extend_jpg /
frame_cnt, 1 );
    Serial.print("Bad    jpg % ");
Serial.println( 100.0 * bad_jpg / frame_cnt,
5 );

    Serial.printf("Writng the index, %d
frames\n", frame_cnt);

    logfile.printf("Recorded %5d frames in
%5d seconds\n", frame_cnt, elapsedms /
1000);
    logfile.printf("File size is %u bytes\n",
movi_size + 12 * frame_cnt + 4);
    logfile.printf("Adjusted FPS is %5.2f\n",
fRealFPS);
    logfile.printf("Max data rate is %lu
bytes/s\n", max_bytes_per_sec);
    logfile.printf("Frame duration is %d us\n",
us_per_frame);
    logfile.printf("Average frame length is %d
bytes\n", uVideoLen / frame_cnt);
    logfile.print("Average picture time (ms)
"); logfile.println( 1.0 * totalp / frame_cnt);
    logfile.print("Average write time (ms)   ");
logfile.println( 1.0 * totalw / frame_cnt );
    logfile.print("Normal jpg % ");
logfile.println( 100.0 * normal_jpg /
frame_cnt, 1 );
    logfile.print("Extend jpg % ");
logfile.println( 100.0 * extend_jpg /
frame_cnt, 1 );
    logfile.print("Bad    jpg % ");
logfile.println( 100.0 * bad_jpg / frame_cnt,
5 );

    logfile.printf("Writng the index, %d
frames\n", frame_cnt);

    avifile.seek( current_end , SeekSet);

    idxfile.close();

    size_t i1_err = avifile.write(idx1_buf, 4);

    print_quartet(frame_cnt * 16, avifile);

    idxfile = SD_MMC.open("/idx.tmp", "r");

    if (idxfile)  {
    }  else  {
      Serial.println("Could not open index
file");
      logfile.println("Could not open index
file");
      major_fail();
    }

    char * AteBytes;
    AteBytes = (char*) malloc (8);

    for (int i = 0; i < frame_cnt; i++) {
      size_t res = idxfile.readBytes( AteBytes,
8);
      size_t i1_err = avifile.write(dc_buf, 4);
      size_t i2_err = avifile.write(zero_buf, 4);
      size_t i3_err = avifile.write((uint8_t
*)AteBytes, 8);
    }

    free(AteBytes);

    idxfile.close();
    avifile.close();

    int xx = remove("/idx.tmp");
  }

  Serial.println("---");  logfile.println("---");

  time_in_sd += (millis() - start);

  Serial.println("");
  time_total = millis() - startms;
  Serial.printf("waiting for cam %10dms,
%4.1f%%\n", wait_for_cam , 100.0 *
wait_for_cam  / time_total);
  Serial.printf("Time in camera  %10dms,
%4.1f%%\n", time_in_camera, 100.0 *
time_in_camera / time_total);
  Serial.printf("waiting for sd  %10dms,
%4.1f%%\n", delay_wait_for_sd , 100.0 *
delay_wait_for_sd  / time_total);
  Serial.printf("Time in sd      %10dms,
%4.1f%%\n", time_in_sd    , 100.0 *
time_in_sd     / time_total);
  Serial.printf("web (core 1)    %10dms,
%4.1f%%\n", time_in_web1  , 100.0 *
time_in_web1   / time_total);
  Serial.printf("web (core 0)    %10dms,
%4.1f%%\n", time_in_web2  , 100.0 *
time_in_web2   / time_total);
  Serial.printf("time total      %10dms,
%4.1f%%\n", time_total    , 100.0 *
time_total     / time_total);
```

```
  logfile.printf("waiting for cam %10dms,
%4.1f%%\n", wait_for_cam , 100.0 *
wait_for_cam  / time_total);
  logfile.printf("Time in camera  %10dms,
%4.1f%%\n", time_in_camera, 100.0 *
time_in_camera / time_total);
  logfile.printf("waiting for sd  %10dms,
%4.1f%%\n", delay_wait_for_sd , 100.0 *
delay_wait_for_sd  / time_total);
  logfile.printf("Time in sd      %10dms,
%4.1f%%\n", time_in_sd    , 100.0 *
time_in_sd     / time_total);
  logfile.printf("web (core 1)    %10dms,
%4.1f%%\n", time_in_web1  , 100.0 *
time_in_web1   / time_total);
  logfile.printf("web (core 0)    %10dms,
%4.1f%%\n", time_in_web2  , 100.0 *
time_in_web2   / time_total);
  logfile.printf("time total      %10dms,
%4.1f%%\n", time_total    , 100.0 *
time_total     / time_total);

  logfile.flush();

}

#include "time.h"

#define _HTTP_Method_H_

typedef enum {
  jHTTP_GET     = 0b00000001,
  jHTTP_POST    = 0b00000010,
  jHTTP_DELETE  = 0b00000100,
  jHTTP_PUT     = 0b00001000,
  jHTTP_PATCH   = 0b00010000,
  jHTTP_HEAD    = 0b00100000,
  jHTTP_OPTIONS = 0b01000000,
  jHTTP_ANY     = 0b01111111,
} HTTPMethod;

#include <WiFi.h>
#include <HTTPClient.h>
HTTPClient http;

time_t now;
struct tm timeinfo;

char the_page[4000];

static esp_err_t capture_handler(httpd_req_t
*req) {

  long start = millis();

  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  char fname[100];
  int file_number = 0;

  file_number++;

  sprintf(fname, "inline;
filename=capture_%d.jpg", file_number);

  if (fb_next == NULL) {
    fb = get_good_jpeg(); //
esp_camera_fb_get();
    framebuffer_len = fb->len;
    memcpy(framebuffer, fb->buf,
framebuffer_len);
    esp_camera_fb_return(fb);
  } else {
    fb = fb_next;
    framebuffer_len = fb->len;
    memcpy(framebuffer, fb->buf,
framebuffer_len);
  }

  httpd_resp_set_type(req, "image/jpeg");
  httpd_resp_set_hdr(req, "Content-
Disposition", fname);
  httpd_resp_set_hdr(req, "Access-Control-
Allow-Origin", "*");

  res = httpd_resp_send(req, (const char
*)framebuffer, framebuffer_len);

  time_in_web1 += (millis() - start);
  return res;
}

static esp_err_t index_handler(httpd_req_t
*req) {

  const char index_html[] PROGMEM =
R"rawliteral(<!doctype html>
  <html>
  <head>
  <meta charset="utf-8">
  <meta name="viewport"
content="width=device-width,initial-
scale=1">
  <title>CCTV MONITORING
MENGGUNAKAN ESP32-CAM </title>
  </head>
  <body> <center>
  <button
onclick="hideIframe();stream.src=host+':81/
stream';">Aktifkan Kamera</button>
```

```
  <button
onclick="hideIframe();stream.src='';">Matik
an Kamera</button><br>
  <button
onclick="hideIframe();getMessage(host+'/co
ntrol?record');getRecordState();">Mulai
Merekam</button>
  <button
onclick="hideIframe();clearTimeout(recordT
imer);getMessage(host+'/control?stop');">Be
rhenti Merekam</button><br>
  <select id="command"
onclick="execute();"><option
value=""></option><option
value="/list">List Data</option><option
value="/control?resetfilegroup">Reset
Data</option></select>
  <span id="message"
style="color:red"></span><br><img
id="stream" src=""
crossorigin="anonymous"><br>
  <iframe id="ifr" width="300" height="200"
style="border: 0px solid
black;display:none"></iframe>
  <script>var
host=window.location.origin;var stream =
document.getElementById("stream");var ifr
= document.getElementById("ifr");var
message =
document.getElementById("message");var
command =
document.getElementById("command");var
recordTimer;function execute()
{message.innerHTML="";if
(command.value!="") {hideIframe();if
(command.value=="/list")
{showIframe();ifr.src =
host+command.value;}else{getMessage(hos
t+command.value);}command.value="";}}f
unction getMessage(url)
{fetch(url).then(function(response) {return
response.text();}).then(function(text) {if
(text=="ok")
clearTimeout(recordTimer);message.innerH
TML=text;});}function hideIframe()
{ifr.style.display="none";}function
showIframe()
{ifr.style.display="block";}function
getRecordState() {recordTimer =
setTimeout(function()
{getMessage(host+'/control?message');getRe
cordState();}, 2000);}</script>
  </body>
  </html>)rawliteral";
```

```
  httpd_resp_set_type(req, "text/html");
  httpd_resp_set_hdr(req, "Access-Control-
Allow-Origin", "*");
  httpd_resp_send(req, (const char
*)index_html, strlen(index_html));

  return ESP_OK;
}

String ListFiles() {
  //SD Card
  if(!SD_MMC.begin()){
    Serial.println("Card Mount Failed");
    return "Card Mount Failed";
  }

  fs::FS &fs = SD_MMC;
  File root = fs.open("/");
  if(!root){
    Serial.println("Failed to open directory");
    return "Failed to open directory";
  }

  String list = "";
  File file = root.openNextFile();
  while(file){
    if(!file.isDirectory()){
      String filename=String(file.name());
      list =
"<tr><td>"+String(file.name())+"</td><td
align='right'>"+String(file.size())+"</td><td
><button
onclick=\"location.href='/control?delete="+S
tring(file.name())+"\';\">Delete</button></t
d></tr>"+list;
    }
    file = root.openNextFile();
  }
  if (list=="") list =
"<tr><td>null</td><td>null</td><td></td>
</tr>";
  list="<table style='border: 1px solid
black'><tr><th align='center'>File
Name</th><th
align='center'>Size</th><th></th></tr>"+lis
t+"</table>";

  file.close();
  SD_MMC.end();

  pinMode(4, OUTPUT);
  digitalWrite(4, LOW);

  return list;
}
```

```
String DeleteFile(String filename) {
 //SD Card
 if(!SD_MMC.begin()){
  Serial.println("Card Mount Failed");
  return "Card Mount Failed";
 }

 fs::FS &fs = SD_MMC;
 File file = fs.open(filename);
 String message="";
 if(fs.remove(filename)){
   message=filename + " File deleted";
 } else {
   message=filename + " Delete failed";
 }
 file.close();
 SD_MMC.end();

 pinMode(4, OUTPUT);
 digitalWrite(4, LOW);

 return message;
}


static esp_err_t cmd_handler(httpd_req_t
*req){
   char*  buf;
   size_t buf_len;
   char variable[128] = {0,};
   char value[128] = {0,};
   String myCmd = "";

   buf_len =
httpd_req_get_url_query_len(req) + 1;
   if (buf_len > 1) {
      buf = (char*)malloc(buf_len);
      if(!buf){
         httpd_resp_send_500(req);
         return ESP_FAIL;
      }
      if (httpd_req_get_url_query_str(req,
buf, buf_len) == ESP_OK) {
        if (httpd_query_key_value(buf, "var",
variable, sizeof(variable)) == ESP_OK &&
          httpd_query_key_value(buf, "val",
value, sizeof(value)) == ESP_OK) {
        }
        else {
          myCmd = String(buf);
        }
      }
      free(buf);
   } else {
      httpd_resp_send_404(req);
      return ESP_FAIL;
   }


Feedback="";Command="";cmd="";P1="";P
2="";P3="";P4="";P5="";P6="";P7="";P8="
";P9="";

ReceiveState=0,cmdState=1,strState=1,quest
ionstate=0,equalstate=0,semicolonstate=0;
   if (myCmd.length()>0) {
     myCmd = "?"+myCmd;
     for (int i=0;i<myCmd.length();i++) {
       getCommand(char(myCmd.charAt(i)));
     }
   }

   if (cmd.length()>0) {
     if (cmd=="record") {
        Serial.println("");
        Serial.println("Start recording");
        frame_cnt = 0;
        start_record = 1;
        Feedback="Start Record - done";
     }
     else if (cmd=="stop") {
       start_record = 0;
       Feedback="Stop Record - done";
     }
     else if (cmd=="delete") {

Feedback=DeleteFile(P1)+"<br>"+ListFiles(
);
     }
     else if (cmd=="resetfilegroup") {
       resetfilegroup = true;
       do_eprom_read();
       Feedback="Reset file group - done";
     }
     else if (cmd=="message") {
       Feedback=recordMessage;
     }
     else if (cmd=="restart") {
       ESP.restart();
     }
     else {
       Feedback="Command is not defined";
     }

     const char *resp = Feedback.c_str();
     httpd_resp_set_type(req, "text/html");
     httpd_resp_set_hdr(req, "Access-
Control-Allow-Origin", "*");
```

```
    return httpd_resp_send(req, resp,
strlen(resp));
   }
   else {
    int val = atoi(value);
    sensor_t * s = esp_camera_sensor_get();
    int res = 0;

    if(!strcmp(variable, "framesize")) {
     if(s->pixformat ==
PIXFORMAT_JPEG)
       res = s->set_framesize(s,
(framesize_t)val);
    }
    else if(!strcmp(variable, "quality")) res =
s->set_quality(s, val);
    else if(!strcmp(variable, "contrast")) res
= s->set_contrast(s, val);
    else if(!strcmp(variable, "brightness"))
res = s->set_brightness(s, val);
    else if(!strcmp(variable, "hmirror")) res
= s->set_hmirror(s, val);
    else if(!strcmp(variable, "vflip")) res = s-
>set_vflip(s, val);
    else if(!strcmp(variable, "flash")) {
     ledcAttachPin(4, 4);
     ledcSetup(4, 5000, 8);
     ledcWrite(4,val);
    }
    else if(!strcmp(variable, "recordonce")) {
       recordOnce = val;
       if (val==1) {
        Serial.println("Record once");
        Feedback="Record once - done";
       }
       else {
        Serial.println("Record
continuously");
         Feedback="Record continuously -
done";
       }
    }
    else if(!strcmp(variable, "avilength")) {
     avi_length = val;
     Serial.println("avi_length = " +
String(val));
      Feedback="avi_length = " +
String(val);
    }
    else {
       res = -1;
    }

    if(res){
       return httpd_resp_send_500(req);
    }

    if (Feedback!="") {
      const char *resp = Feedback.c_str();
      httpd_resp_set_type(req, "text/html");
      httpd_resp_set_hdr(req, "Access-
Control-Allow-Origin", "*");
      return httpd_resp_send(req, resp,
strlen(resp));
    }
    else {
      httpd_resp_set_hdr(req, "Access-
Control-Allow-Origin", "*");
      return httpd_resp_send(req, NULL, 0);
    }
   }
}

static esp_err_t list_handler(httpd_req_t
*req) {
 String list = "";
 if (recordMessage!="ok")
  list = "Get error. Recording...";
 else
  list = ListFiles();

 //Serial.println(list);
 const char msg[] PROGMEM =
R"rawliteral(<!doctype html>
 <html>
 <head>
 <meta charset="utf-8">
 <meta name="viewport"
content="width=device-width,initial-
scale=1">
 </head>
 <body>
 %s
 </body>
 </html>)rawliteral";

 sprintf(the_page, msg, list.c_str());

 httpd_resp_set_type(req, "text/html");
 httpd_resp_set_hdr(req, "Access-Control-
Allow-Origin", "*");
 httpd_resp_send(req, the_page,
strlen(the_page));

 return ESP_OK;
}

bool start_streaming = false;
```

```
#define PART_BOUNDARY
"123456789000000000000987654321"

void the_streaming_loop (void*
pvParameter);

static esp_err_t stream_handler(httpd_req_t
*req) {

  long start = millis();

  Serial.print("stream_handler, core ");
Serial.print(xPortGetCoreID());
  Serial.print(", priority = ");
Serial.println(uxTaskPriorityGet(NULL));

  start_streaming = true;

  xTaskCreatePinnedToCore(
the_streaming_loop, "the_streaming_loop",
8000, req, 1, &the_streaming_loop_task, 0);

  if ( the_streaming_loop_task == NULL ) {
    //vTaskDelete( xHandle );
    Serial.printf("do_the_steaming_task failed
to start! %d\n", the_streaming_loop_task);
  }

  time_in_web1 += (millis() - start);

  while (start_streaming == true) {
    delay(1000);
    //Serial.print("z");
  }
  Serial.println("Streaming done");
}

void the_streaming_loop (void*
pvParameter) {
  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];

  long start = millis();

  Serial.print("\n\nlow prio stream_handler
task, core ");
Serial.print(xPortGetCoreID());
  Serial.print(", priority = ");
Serial.println(uxTaskPriorityGet(NULL));

  req = (httpd_req_t *) pvParameter;

  Serial.println("Starting the streaming");

  res = httpd_resp_set_type(req,
_STREAM_CONTENT_TYPE);
  if (res != ESP_OK) {
    Serial.printf("after first
httpd_resp_set_type %d\n", res);
    start_streaming = false;
  }

  int streaming_frames = 0;
  long streaming_start = millis();

  while (true) {
    streaming_frames++;

    if (fb_curr == NULL) {
      fb = get_good_jpeg();
//esp_camera_fb_get();
      if (!fb) {
        Serial.println("Stream - Camera
Capture Failed");
        start_streaming = false;
      }
      framebuffer_len = fb->len;
      memcpy(framebuffer, fb->buf, fb->len);
      esp_camera_fb_return(fb);

    } else {
      fb = fb_curr;
      framebuffer_len = fb->len;
      memcpy(framebuffer, fb->buf, fb->len);
    }

    _jpg_buf_len = framebuffer_len;
    _jpg_buf = framebuffer;

    size_t hlen = snprintf((char *)part_buf,
64, _STREAM_PART, _jpg_buf_len);
    long send_time = millis();

    httpd_resp_set_hdr(req, "Access-Control-
Allow-Origin", "*");
    res = httpd_resp_send_chunk(req, (const
char *)part_buf, hlen);
    if (res != ESP_OK) {
      //Serial.printf("Stream error - 1st send
chunk %d\n",res);
      start_streaming = false;
    }

    res = httpd_resp_send_chunk(req, (const
char *)_jpg_buf, _jpg_buf_len);
    if (res != ESP_OK) {
```

```
      //Serial.printf("Stream error - 2nd send
chunk %d\n",res);
      start_streaming = false;
    }

    res = httpd_resp_send_chunk(req,
_STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
    if (res != ESP_OK) {
      //Serial.printf("Stream error - 3rd send
chunk %d\n", res);
      start_streaming = false;
    }

    if (millis() - send_time > stream_delay) {
      stream_delay = stream_delay * 1.5;
    }

    time_in_web2 += (millis() - start);

    if (streaming_frames % 100 == 10) {
      if (Lots_of_Stats)
Serial.printf("Streaming at %3.3f fps\n",
(float)1000 * streaming_frames / (millis() -
streaming_start));
    }

    delay(stream_delay);
    start = millis();

    if (start_streaming == false) {
      Serial.println("Deleting the streaming
task");
      delay(100);
      vTaskDelete(the_streaming_loop_task);
    }
  }  // stream forever
}

void startCameraServer() {
  httpd_config_t config =
HTTPD_DEFAULT_CONFIG();

  Serial.print("http task prio: ");
Serial.println(config.task_priority);

  httpd_uri_t index_uri = {
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = index_handler,
    .user_ctx  = NULL
  };
  httpd_uri_t capture_uri = {
    .uri       = "/capture",
    .method    = HTTP_GET,
    .handler   = capture_handler,
    .user_ctx  = NULL
  };
  httpd_uri_t stream_uri = {
    .uri       = "/stream",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
  };
  httpd_uri_t cmd_uri = {
     .uri       = "/control",
     .method    = HTTP_GET,
     .handler   = cmd_handler,
     .user_ctx  = NULL
  };
  httpd_uri_t list_uri = {
    .uri       = "/list",
    .method    = HTTP_GET,
    .handler   = list_handler,
    .user_ctx  = NULL
  };

  ra_filter_init(&ra_filter, 20);
  Serial.println("");
  Serial.printf("Starting web server on port:
'%d'\n", config.server_port);  //Server Port
  if (httpd_start(&camera_httpd, &config) ==
ESP_OK) {
     //註冊自訂網址路徑對應執行的函式
    httpd_register_uri_handler(camera_httpd,
&index_uri);
    httpd_register_uri_handler(camera_httpd,
&capture_uri);
    httpd_register_uri_handler(camera_httpd,
&stream_uri);
    httpd_register_uri_handler(camera_httpd,
&cmd_uri);
    httpd_register_uri_handler(camera_httpd,
&list_uri);
  }

  config.server_port += 1;  //Stream Port
  config.ctrl_port += 1;    //UDP Port

  Serial.printf("Starting stream server on
port: '%d'\n", config.server_port);
  if (httpd_start(&stream_httpd, &config) ==
ESP_OK) {
     httpd_register_uri_handler(stream_httpd,
&stream_uri);
  }
  Serial.println("Camera http started");
  Serial.println("");
}
```

```
void stopCameraServer() {
  httpd_stop(camera_httpd);
}

void setup() {

WRITE_PERI_REG(RTC_CNTL_BROWN
_OUT_REG, 0);

  Serial.begin(115200);
  //Serial.setDebugOutput(true);
  Serial.println();

  pinMode(33, OUTPUT);            // little red
led on back of chip
  digitalWrite(33, LOW);          // turn on the
red LED on the back of chip

  pinMode(4, OUTPUT);             // Blinding
Disk-Avtive Light
  digitalWrite(4, LOW);           // turn off

  WiFi.mode(WIFI_AP_STA);

  WiFi.begin(ssid, password);

  delay(1000);
  Serial.println("");
  Serial.print("Connecting to ");
  Serial.println(ssid);

  long int StartTime=millis();
  while (WiFi.status() !=
WL_CONNECTED)
  {
     delay(500);
     if ((StartTime+5000) < millis()) break;
  }

  if (WiFi.status() != WL_CONNECTED) {
    WiFi.begin(ssid, password);

    StartTime=millis();
    while (WiFi.status() !=
WL_CONNECTED)
    {
       delay(500);
       if ((StartTime+5000) < millis()) break;
    }
  }

  if (WiFi.status() == WL_CONNECTED) {

WiFi.softAP((WiFi.localIP().toString()+"_"
+(String)apssid).c_str(), appassword);
    Serial.println("");
    Serial.println("STAIP address: ");
    Serial.println(WiFi.localIP());
  }
  else {

WiFi.softAP((WiFi.softAPIP().toString()+"_
"+(String)apssid).c_str(), appassword);
  }

  Serial.println("");
  Serial.println("APIP address: ");
  Serial.println(WiFi.softAPIP());
  Serial.println("");

  Serial.println("                              ");
  Serial.println("----------------------------------
---");
  Serial.printf("ESP32-CAM monitoring
%s\n", vernum);
  Serial.println("----------------------------------
---");

  Serial.print("setup, core ");
Serial.print(xPortGetCoreID());
  Serial.print(", priority = ");
Serial.println(uxTaskPriorityGet(NULL));

  esp_reset_reason_t reason =
esp_reset_reason();

  logfile.print("--- reboot ------ because: ");
  Serial.print("--- reboot ------ because: ");

  switch (reason) {
    case ESP_RST_UNKNOWN :
Serial.println("ESP_RST_UNKNOWN");
logfile.println("ESP_RST_UNKNOWN");
break;
    case ESP_RST_POWERON :
Serial.println("ESP_RST_POWERON");
logfile.println("ESP_RST_POWERON");
break;
    case ESP_RST_EXT :
Serial.println("ESP_RST_EXT");
logfile.println("ESP_RST_EXT"); break;
    case ESP_RST_SW :
Serial.println("ESP_RST_SW");
logfile.println("ESP_RST_SW"); break;
    case ESP_RST_PANIC :
Serial.println("ESP_RST_PANIC");
logfile.println("ESP_RST_PANIC"); break;
    case ESP_RST_INT_WDT :
Serial.println("ESP_RST_INT_WDT");
```

```
logfile.println("ESP_RST_INT_WDT");
break;
    case ESP_RST_TASK_WDT :
Serial.println("ESP_RST_TASK_WDT");
logfile.println("ESP_RST_TASK_WDT");
break;
    case ESP_RST_WDT :
Serial.println("ESP_RST_WDT");
logfile.println("ESP_RST_WDT"); break;
    case ESP_RST_DEEPSLEEP :
Serial.println("ESP_RST_DEEPSLEEP");
logfile.println("ESP_RST_DEEPSLEEP");
break;
    case ESP_RST_BROWNOUT :
Serial.println("ESP_RST_BROWNOUT");
logfile.println("ESP_RST_BROWNOUT");
break;
    case ESP_RST_SDIO :
Serial.println("ESP_RST_SDIO");
logfile.println("ESP_RST_SDIO"); break;
    default  : Serial.println("Reset resaon");
logfile.println("ESP ???"); break;
  }

  Serial.println("Reading the eprom  ...");
  do_eprom_read();

  // SD camera init
  Serial.println("Mounting the SD card ...");
  esp_err_t card_err = init_sdcard();
  if (card_err != ESP_OK) {
    Serial.printf("SD Card init failed with
error 0x%x", card_err);
    major_fail();
    return;
  }

  devstr.toCharArray(devname,
devstr.length()+1);          // name of your
camera for mDNS, Router, and filenames

  Serial.println("Try to get parameters from
config.txt ...");
  read_config_file();

  char logname[50];
  sprintf(logname, "/%s%d.999.txt",
devname, file_group);
  Serial.printf("Creating logfile %s\n",
logname);
  logfile = SD_MMC.open(logname,
FILE_WRITE);

  if (!logfile) {
    Serial.println("Failed to open logfile for
writing");
  }

  Serial.println("Setting up the camera ...");
  config_camera();

  Serial.println("Checking SD for available
space ...");
  delete_old_stuff();

  digitalWrite(33, HIGH);

  startCameraServer();

  framebuffer = (uint8_t*)ps_malloc(1024 *
1024); // buffer to store a jpg in motion //
needs to be larger for big frames from
ov5640

  Serial.println("Creating
the_camera_loop_task");

  wait_for_sd = xSemaphoreCreateMutex();
  sd_go = xSemaphoreCreateMutex();

  xSemaphoreTake( wait_for_sd,
portMAX_DELAY );   // will be "given"
when sd write is done
  xSemaphoreTake( sd_go,
portMAX_DELAY );         // will be "given"
when sd write should start

  // prio 3 - higher than the camera loop(),
and the streaming
  xTaskCreatePinnedToCore(
the_camera_loop, "the_camera_loop", 3000,
NULL, 3, &the_camera_loop_task, 0); //
prio 3, core 0

  delay(100);

  // prio 4 - higher than the cam_loop(), and
the streaming
  xTaskCreatePinnedToCore( the_sd_loop,
"the_sd_loop", 2000, NULL, 4,
&the_sd_loop_task, 1);  // prio 4, core 1

  delay(200);

  boot_time = millis();

  const char *strdate = ctime(&now);
  logfile.println(strdate);
}
```

```
void the_sd_loop (void* pvParameter) {

  Serial.print("the_sd_loop, core ");
Serial.print(xPortGetCoreID());
  Serial.print(", priority = ");
Serial.println(uxTaskPriorityGet(NULL));

  while (1) {
    xSemaphoreTake( sd_go,
portMAX_DELAY );
    another_save_avi( fb_curr);
    xSemaphoreGive( wait_for_sd );
  }
}

void the_camera_loop (void* pvParameter)
{

  Serial.print("the loop, core ");
Serial.print(xPortGetCoreID());
  Serial.print(", priority = ");
Serial.println(uxTaskPriorityGet(NULL));

  frame_cnt = 0;
  start_record = 0;

  delay(500);

  while (1) {

    if (frame_cnt == 0 && start_record == 0)
{

      Serial.println("ok");
      recordMessage = "ok";

      we_are_already_stopped = 1;
      delay(2000);
    } else if (frame_cnt == 0 && start_record
== 1) {

      Serial.println("Ready to start");
      recordMessage = "Ready to start";

      we_are_already_stopped = 0;

      delete_old_stuff();


      frame_cnt++;

      long wait_for_cam_start = millis();
      fb_curr = get_good_jpeg();
      wait_for_cam += millis() -
wait_for_cam_start;

      start_avi();

      wait_for_cam_start = millis();
      fb_next = get_good_jpeg();
      wait_for_cam += millis() -
wait_for_cam_start;

      xSemaphoreGive( sd_go );

      digitalWrite(33, frame_cnt % 2);

    } else if ( (frame_cnt > 0 && start_record
== 0) ||  millis() > (avi_start_time +
avi_length * 1000)) { // end the avi

      Serial.println("End the Avi");
      recordMessage = "End the Avi";

      xSemaphoreTake( wait_for_sd,
portMAX_DELAY );
      esp_camera_fb_return(fb_curr);

      frame_cnt++;
      fb_curr = fb_next;
      fb_next = NULL;

      xSemaphoreGive( sd_go );                //
save final frame of movie

      digitalWrite(33, frame_cnt % 2);

      xSemaphoreTake( wait_for_sd,
portMAX_DELAY );    // wait for final
frame of movie to be written

      esp_camera_fb_return(fb_curr);
      fb_curr = NULL;

      end_avi();                              // end the
movie

      digitalWrite(33, HIGH);          // light off

      avi_end_time = millis();

      float fps = 1.0 * frame_cnt /
((avi_end_time - avi_start_time) / 1000) ;

      Serial.printf("End the avi at %d.  It was
%d frames, %d ms at %.2f fps...\n", millis(),
frame_cnt, avi_end_time, avi_end_time -
avi_start_time, fps);
```

```
    logfile.printf("End the avi at %d.  It was
%d frames, %d ms at %.2f fps...\n", millis(),
frame_cnt, avi_end_time, avi_end_time -
avi_start_time, fps);

    frame_cnt = 0;

    if (recordOnce == true) {
      start_record = 0;
    }
   } else if (frame_cnt > 0 && start_record
!= 0) {  // another frame of the avi

    //Serial.println("Another frame");
    recordMessage = "Recording...";

    current_frame_time = millis();
    if (current_frame_time - last_frame_time
< frame_interval) {
      delay(frame_interval -
(current_frame_time - last_frame_time));
// delay for timelapse
    }
    last_frame_time = millis();

    frame_cnt++;

    long delay_wait_for_sd_start = millis();
    xSemaphoreTake( wait_for_sd,
portMAX_DELAY );            // make sure sd
writer is done
    delay_wait_for_sd += millis() -
delay_wait_for_sd_start;

    esp_camera_fb_return(fb_curr);

    fb_curr = fb_next;

    xSemaphoreGive( sd_go );

    long wait_for_cam_start = millis();
    fb_next = get_good_jpeg();
    wait_for_cam += millis() -
wait_for_cam_start;

    digitalWrite(33, frame_cnt % 2);

    if (frame_cnt % 100 == 10 ) {     // print
some status every 100 frames
      if (frame_cnt == 10) {
        bytes_before_last_100_frames =
movi_size;
        time_before_last_100_frames =
millis();
        most_recent_fps = 0;
        most_recent_avg_framesize = 0;
      } else {


        if (Lots_of_Stats) {
          Serial.printf("So far: %04d frames, in
%6.1f seconds, for last 100 frames: avg
frame size %6.1f kb, %.2f fps ...\n",
frame_cnt, 0.001 * (millis() -
avi_start_time), 1.0 / 1024  *
most_recent_avg_framesize,
most_recent_fps);
          logfile.printf("So far: %04d frames,
in %6.1f seconds, for last 100 frames: avg
frame size %6.1f kb, %.2f fps ...\n",
frame_cnt, 0.001 * (millis() -
avi_start_time), 1.0 / 1024  *
most_recent_avg_framesize,
most_recent_fps);
        }

        total_delay = 0;

        bytes_before_last_100_frames =
movi_size;
        time_before_last_100_frames =
millis();
      }
    }
   }
  }
}

void loop() {

  if (reboot_now == true) {
    delay(2000);
    major_fail();
  }

  delay(200);
}

void getCommand(char c)
{
  if (c=='?') ReceiveState=1;
  if ((c==' ')||(c=='\r')||(c=='\n'))
ReceiveState=0;

  if (ReceiveState==1)
  {
    Command=Command+String(c);

    if (c=='=') cmdState=0;
```

```
    if (c==';') strState++;

    if
((cmdState==1)&&((c!='?')||(questionstate=
=1))) cmd=cmd+String(c);
    if
((cmdState==0)&&(strState==1)&&((c!='=')
||(equalstate==1))) P1=P1+String(c);
    if
((cmdState==0)&&(strState==2)&&(c!=';'))
P2=P2+String(c);
    if
((cmdState==0)&&(strState==3)&&(c!=';'))
P3=P3+String(c);
    if
((cmdState==0)&&(strState==4)&&(c!=';'))
P4=P4+String(c);
    if
((cmdState==0)&&(strState==5)&&(c!=';'))
P5=P5+String(c);
    if
((cmdState==0)&&(strState==6)&&(c!=';'))
P6=P6+String(c);
    if
((cmdState==0)&&(strState==7)&&(c!=';'))
P7=P7+String(c);
    if
((cmdState==0)&&(strState==8)&&(c!=';'))
P8=P8+String(c);
    if
((cmdState==0)&&(strState>=9)&&((c!=';')|
|(semicolonstate==1))) P9=P9+String(c);

    if (c=='?') questionstate=1;
    if (c=='=') equalstate=1;
    if ((strState>=9)&&(c==';'))
semicolonstate=1;
  }
}
```