# Code Explanation

First, I would really like to thank and acknowledge Random Nerd Tutorials for giving the resources and tutorials needed to help me develop this script. They make all their tutorials free and also do their best to help out their community and give so many valuble resources for makers. Please check them out!

## Library Imports

Here we, include all the libraries we need for our script. These library enable us to use the ESP32 Camera, write to the microSD card, connect to WiFi, and determine the time using NTP.

```
#include "esp_camera.h"
#include "Arduino.h"
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include <SPI.h>               // SD Card ESP32
#include "soc/soc.h"           // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
```

## Script Parameters and Variables

This is where we define our WiFi credentials. Make sure that you have changed those parameters so that the ESP32 CAM can properly connect to a WiFi network. This is important for determining the time.

```
// === wifi credentials ===
// replace with your network credentials
const char* WIFI_SSID = "CHANGE_TO_YOUR_WIFI_SSID";
const char* WIFI_PASS = "CHANGE_TO_YOUR_WIFI_PASSWORD";
```

If you have a need to change the pin assignment for the HC-SR04, feel free to change it here, but it is not reccomended because the SD card reader will be using other pins.

```
// === peripherals pin assignment ===
const int ultrasonicTrigPin = 13;
const int ultrasonicEchoPin = 12;
```

This here determines the time the ESP32 CAM should go into deep sleep. In this script, it will be sleeping for 1,800 seconds or 30 minutes. Deep sleep helps reduce energy consumption when nothing is being done. You can feel free to change the time to sleep value.

```
// === deep sleep ===
#define uS_TO_S_FACTOR 1000000 /* conversion factor for usec to sec */
#define TIME_TO_SLEEP 1800 /* TIME ESP32 will go to sleep in sec */
```

Here we define all our other parameters used for determining the fullness level, figuring out where to save our fullness data, saving the datetime stamp, creating the NTP Client object, and defining all the pins for the AI THINKER ESP32 CAM model.

```
// === other parameters ===
const int binHeight = 100; // units: cm, used to calculate fullness
const int ultraMinRange = 2; //units: cm, min. limit of accurate HC-SR04 reading
const int ultraMaxRange = 400; //units: cm, max. limit of accurate HC-SR04 reading
const char* datalogFile = "/data.csv";

// === system variables ===
long duration; // units: microseconds, for HC-SR04
int distance; // units:cm, for HC-SR04
int fullness; //units:cm, the bin fullness
String datetimeStamp;
String path;
String dataMessage;

// Define NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);

// === camera pin definitions ===
#define FLASH_LED_PIN 4

//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
```

## configInitCamera() function

Here we set up all the camera connections and also any configuration settings for the AI THINKER ESP32 CAM model.

```
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;
```

This is where we define the resolution of our image quality. You can change these settings to help you increase or reduce the image quality. Please read comments for more details. You can specify the jpeg quality and also the frame size such as UXGA, SGXGA, SVGA, etc.

```
//=== init with high specs to pre-allocate larger buffers ===
if(psramFound()){
  Serial.println("PSRAM found");
  config.frame_size = FRAMESIZE_UXGA;
  config.jpeg_quality = 20;  //0-63 lower number means higher quality
  config.fb_count = 2;
} else {
  Serial.println("PSRAM not found");
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 15;  //0-63 lower number means higher quality
  config.fb_count = 1;
}

// === Init Camera ===
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Camera init failed with error 0x%x", err);
  return;
```

```
  }

  // Drop down frame size for higher initial frame rate
  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAMESIZE_UXGA);  // UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA
```

## takePhoto() function

I think the comments should cover the explanation here, please just read the comments in the function.

## fullnessRead() function

I also think the comments should cover the explanation, please read the comments int the function.

## getDateTime() function

This really just uses the NTPClient.h library we imported before. We are using the `timeClient` object to interact with the NTP server and fetch the time. Then, if you notice we add a `-25200` offset so that we can get the datetime in Pacific Standard Time (PST). You can change this offset to reflect your timezone. The output format of the datetime stamp will look something like this `2018-04-30T16:00:13Z`. This datetime stamp will be used as the image file names when a photo is taken and also for the CSV file that will save the fullness measurements.

```
void getDateTime() {
  timeClient.begin();
  timeClient.setTimeOffset(-25200); // offset for PST
  while(!timeClient.update()) {
    timeClient.forceUpdate();
  }
  datetimeStamp = timeClient.getFormattedDate(); //format: 2018-04-30T16:00:13Z
  datetimeStamp.replace(":","-"); // get rid of colons

  //=== end the client ===
  timeClient.end();
}
```

## writeFile() function

This is a function that writes a file to an SD card. You can see that it takes the `path` parameter, which is the path to the file you want to save in the SD card, and also the `message` parameter specifies what message you want to save in the file. This will write a new file if the file doesn't exist or overwrite an existing one if the same file already exists. We already defined a variable to write to the files `data.csv`

```
void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
```

```
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)){
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
}
```

## appendFile() function

This is very similar to the writeFile() function. However, instead of writing a new file everytime, it simply appends the message to an existing file. So you must make sure the file you are appending to already exists. We already definied a variable to append to the file `data.csv`.

```
void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)){
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
}
```

## sdSetup() function

The very first part of this function simply mounts the SD card. Next once, the card is mounted successfully, we detect the card. Most SD cards nowadays are MMC, which stands for MultiMediaCard. So if you notice I do `SD_MMC.begin("/sdcard",true)` in the if statement. This will set the SD card up in 1-bit SD bus mode as opposed to 4-bit SD bus mode. This means that the transfer rate for data will be slower, but at the same time it frees up GPIO pins 12 and 13 for us to use with the ultrasonic sensor. For more details about SD bus modes please refer to the *Additional Resources* section below. The rest of this code simply creates the `data.csv` file if it doesn't exist already.

```
void sdSetup() {
  // === sd card setup ===
  // Card Mounting
  Serial.println("Starting SD Card");
  if(!SD_MMC.begin("/sdcard", true)){
    Serial.println("SD Card Mount Failed");
    return;
```

```
  }

  //Detect Card
  uint8_t cardType = SD_MMC.cardType();
  if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
    return;
  }

  // If the data.txt file doesn't exist
  // Create a file on the SD card and write the data labels
  File file = SD_MMC.open(datalogFile);
  if(!file) {
    file.close();
    Serial.println("File doesn't exist");
    writeFile(SD_MMC, datalogFile, "Datetimestamp,fullness(cm)\n");
    return;
  }
  else {
    Serial.println("File already exists");
    file.close();
  }
}
```

## setup() function

The setup function first sets up the serial monitor so that you can debug anything by opening the serial monitor. Then we setup everything we need such as the ultrasonic sensor, the SD card, it connects to WiFi, and also setups the camera.

```
void setup() {
  Serial.begin(115200);
  Serial.println("Setup has started...");

  // === deep sleep setup ===
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) + " Seconds");

  // === ultrasonic setup ===
  pinMode(ultrasonicTrigPin, OUTPUT);
  pinMode(ultrasonicEchoPin, INPUT);

  // === sd card setup ===
  sdSetup();

  // === wifi setup ===
  Serial.print("Connecting to ");
  Serial.println(WIFI_SSID);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // === camera setup ===
  pinMode( FLASH_LED_PIN, OUTPUT); // Setup flash
  configInitCamera(); // Setup camera

  Serial.println("Setup finished");
  Serial.println();

}
```

## loop() function

This function is really the main function that helps us run all the tasks that we want to do. It takes a photo, gets the distance reading and writes everything to the SD card. Once everything is written to the SD, we go into deep sleep mode. This allows the device to reduce its energy consumption which is very useful when running on a battery source.

```
void loop() {

  // === take a photo ===
  digitalWrite(FLASH_LED_PIN,HIGH);
  takePhoto(); // will also automatically update the datetimeStamp
  digitalWrite(FLASH_LED_PIN,LOW);

  // === get distance reading ===
  fullnessRead();

  // === write data to sd card ===
  //update the datetime stamp
  getDateTime();
  //create data message
  dataMessage = datetimeStamp + "," + String(fullness) + "\n";
  Serial.print("Writing ");
  Serial.print(dataMessage);
  Serial.println();

  // convert String to char*
  int str_len = dataMessage.length()+1;
  char charBuf[str_len];
  dataMessage.toCharArray(charBuf,str_len);
```

```
  // write to sd card
  appendFile(SD_MMC, datalogFile, charBuf);

  // === sleep timer ===
  Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();

}
```

# Resources

- https://randomnerdtutorials.com/esp32-ntp-client-date-time-arduino-ide/
- https://randomnerdtutorials.com/telegram-esp32-cam-photo-arduino/
- https://randomnerdtutorials.com/esp32-cam-take-photo-save-microsd-card/
- https://techtutorialsx.com/2020/09/06/esp32-writing-file-to-sd-card/
- https://www.instructables.com/Select-SD-Interface-for-ESP32/
- https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/