IoT Lab

Over The Air (OTA) programming

Table of Contents

1.1 Introduction

OTA programming enables updating - uploading a new program to ESP32 **via Wi-Fi** instead of forcing the user to connect the ESP32 to a computer via USB to update.

The OTA functionality is extremely useful if there is **no physical access** to the ESP module. This reduces the time spent updating each ESP module during maintenance.

An important feature of OTA is that a single central location can send an update to **multiple ESP**s sharing the same network.

The only **downside** is that you have to add **extra code for OTA** to every program you download, so you can use OTA in the next update.

1.1.1 Flash memory partitions

Before moving on to programming, we will study the flash memory partitions of an ESP32.
Flash memory is divided into **several logical partitions** to store various components. The typical way to do this is shown in the figure.

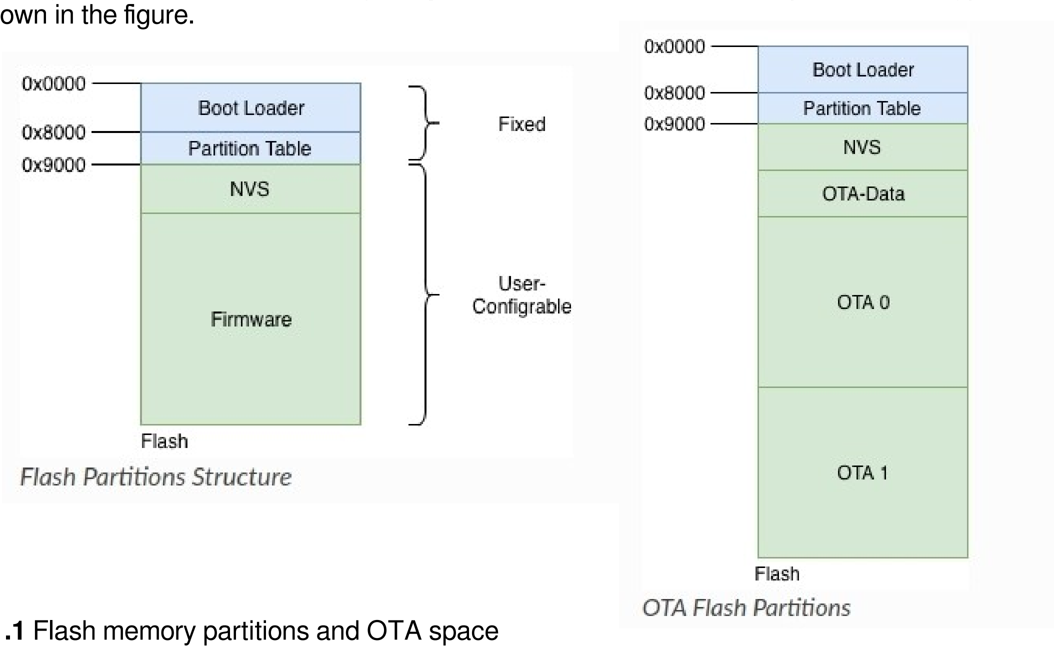


Figure 1.1 Flash memory partitions and OTA space

As can be seen in the figure above, the structure is static up to the flash address `0x9000`.

The **first part** of the flash contains the **boot code**, which is immediately **followed by** the **partition table**.

The partition table then stores how the **rest of the flash** should be interpreted.Typically an installation will have **at least 1 NVS** (**Non Volatile Storage – WiFi credentials** , ..) partition and one partition for **user code**

1.1.2 Mécanisme OTA

For code upgrades, an **active-passive partition** scheme is used. **Two flash partitions** are reserved for the 'firmware' component, as shown in the above figure. The **OTA-Data partition** remembers which of these is the **active partition**.

Typical state changes that occur in the OTA code upgrade workflow are shown in Figure 1.2.

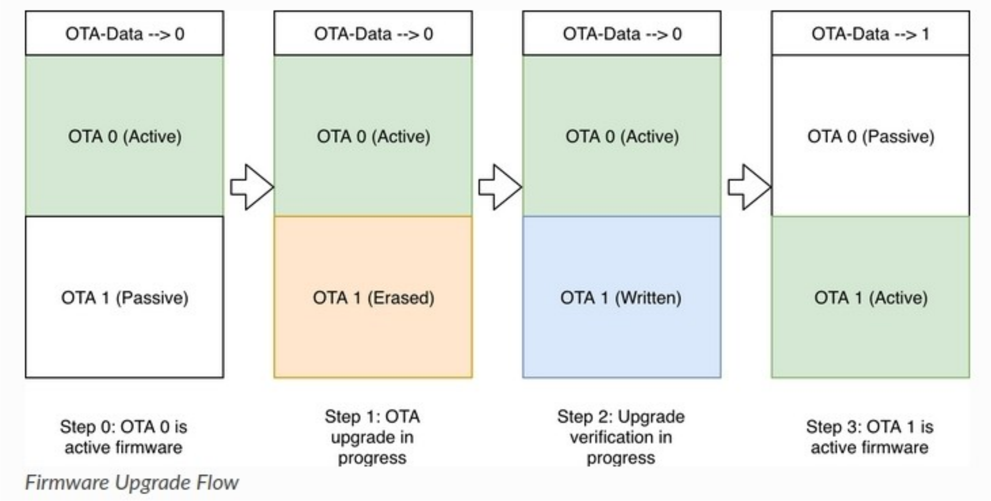


Figure 1.2 Flow of **code upgrade** in flash memory

1. **Step 0**: OTA 0 is the active code. The OTA data partition stores this information as can be seen.
2. **Step 1**: The code upgrade process begins. The passive partition is identified, erased and a new code is being written to OTA partition 1.
3. **Step 2**: Code upgrade is fully written and verification is in progress.
4. **Step 3**: Code upgrade is successful, OTA data partition is updated to show that OTA 1 is now the active partition. On the next boot, the code will boot for this partition.

1.2 Implementation of OTA on ESP32 board by basic OTA

There are **three ways** to implement **OTA** functionality in ESP32.

1. **Basic OTA** - Over-The-Air updates are sent via Arduino IDE or PlatformIO.
2. **Web Updater OTA** - Over-the-air updates are delivered through a web browser.
3. The **WebOTA library** also allows to send the compiled Arduino sketch (`.bin`) directly via the WEB interface.

Each way has its own advantages. You can implement them as needed for your project.

1.2.1 Basic OTA implementation

To get started, download basic OTA firmware through a serial port. This is a mandatory step to be able to perform future updates - downloads via WiFi. In the next phase, you can upload new programs to ESP32 from Arduino IDE via air - WiFi with an IP address.

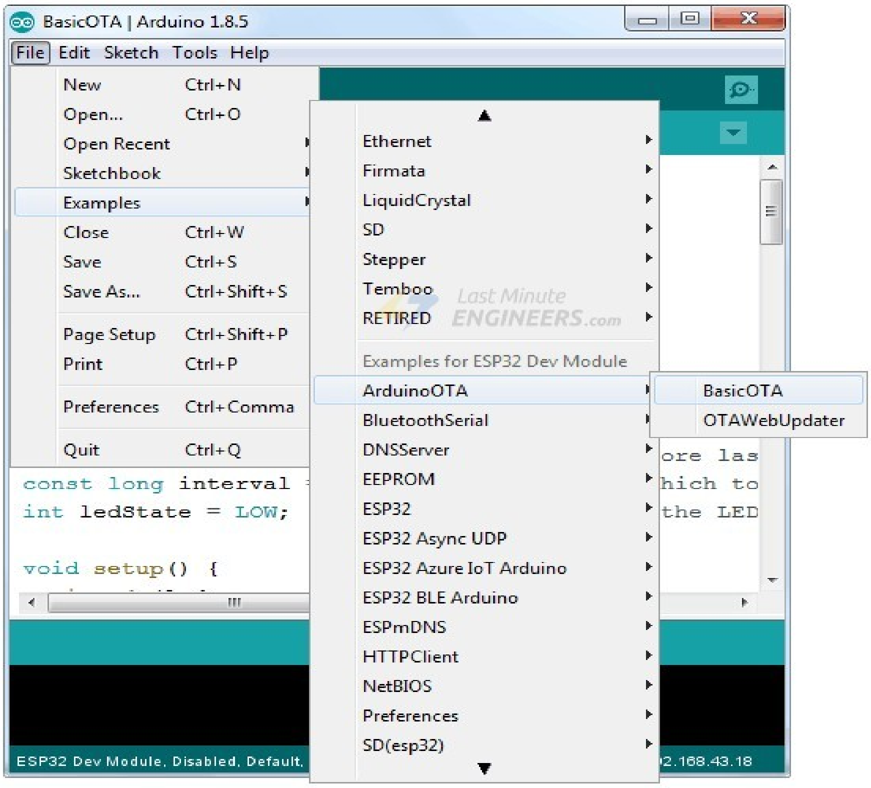


Figure 1.3 Basic OTA code selection

Before downloading the code, you need to provide some changes to make it work. You need to modify the following **two variables** with your **network credentials**, so that ESP32 can establish a connection with the existing network.

```
const char* ssid = "..........";
const char* password = "..........";
```

Once you are done, download the code via USB cable.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

No
w open the serial monitor at a baud rate of `115200`. If all is well, the **dynamic IP address** obtained from your router will be displayed. **Write it down**.

1.2.2 Download a new code via WiFi

Now let's download a **new program**.

It is necessary to **add the code for OTA** in each sketch you upload. Otherwise, **you will lose OTA capability** and will not be able to perform future over-the-air downloads. It is therefore recommended that you modify the code above to include your new code.

As an example, we'll include a simple **Blink** sketch in the basic OTA code. Remember to modify the **`ssid`** and **`password`** variables with your network credentials.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";
//variables for blinking an LED with Millis
const int led = 22; // ESP32 Lolin D32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000); ESP.restart();
  }

  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";
      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  ArduinoOTA.handle();
  //loop to blink without delay
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
  // save the last time you blinked the LED
  previousMillis = currentMillis;
  // if the LED is off turn it on and vice-versa:
  ledState = not(ledState);
  // set the LED with the ledState of the variable:
  digitalWrite(led,  ledState);
  }
}
```

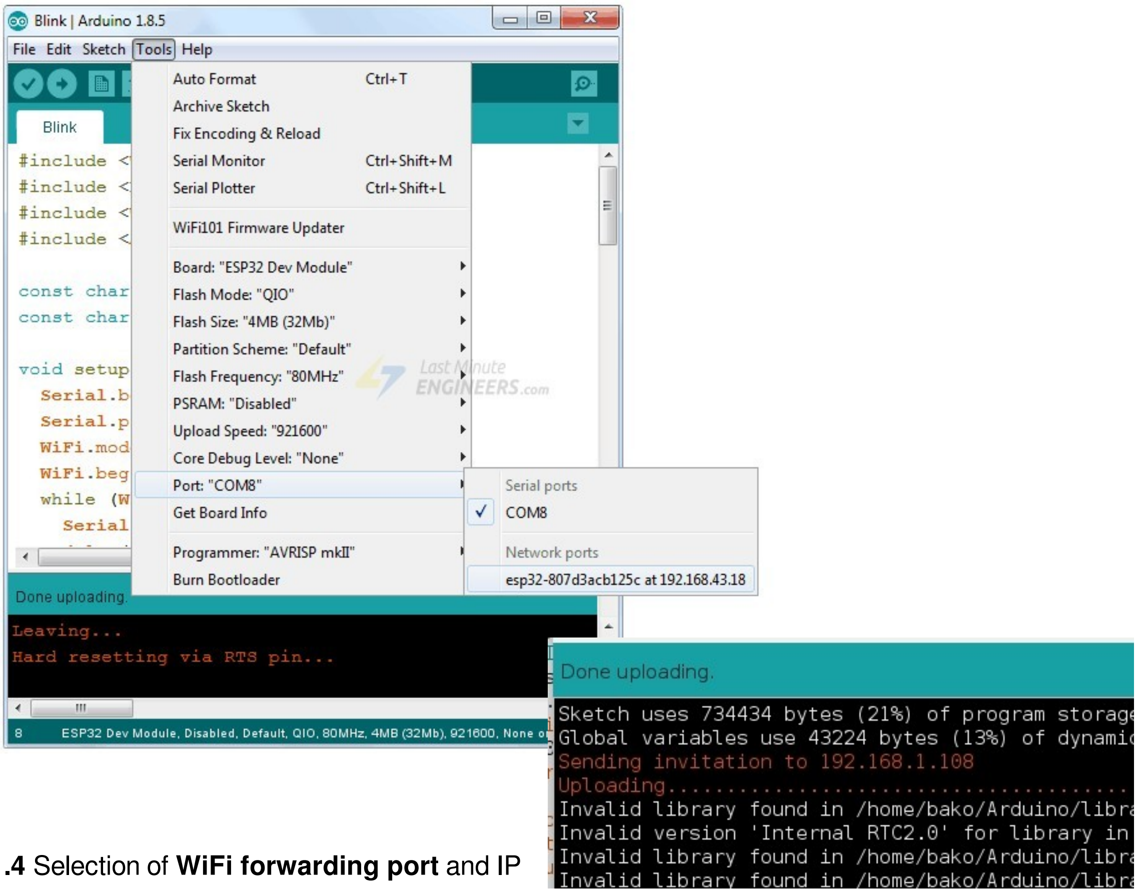
Attention !

In the program above, we didn't use **`delay()`** to make the LED blink, because ESP32 pauses your program during **`delay()`**. If the next OTA request is generated while Arduino is waiting for **`timeout()`**, your program **will miss that request**.

To achieve the delay we used the **timer**: **`currentMillis = millis()`**

Once you have copied the sketch above to your Arduino IDE, navigate to the **Tools->Port** option and you should get the following output: **esp32-xxxxxx at esp_ip_address (LOLIN D32)** for your board

If you can't find it, you may need to restart your IDE.



```
#include <
#include <
#include <
#include <

const char
const char

void setup
  Serial.b
  Serial.p
  WiFi.mod
  WiFi.beg
  while (W
    Serial
```

Board: "ESP32 Dev Module"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default"
Flash Frequency: "80MHz"
PSRAM: "Disabled"
Upload Speed: "921600"
Core Debug Level: "None"
Port: "COM8"
Programmer: "AVRISP mkII"

✓ COM8

esp32-807d3acb125c at 192.168.43.18

Done uploading.

```
Leaving...
Hard resetting via RTS pin...
```

8 ESP32 Dev Module, Disabled, Default, QIO, 80MHz, 4MB (32Mb), 921600, None o

Done uploading.

```
Sketch uses 734434 bytes (21%) of program storage
Global variables use 43224 bytes (13%) of dynamic
Sending invitation to 192.168.1.108
Uploading.....................................
Invalid library found in /home/bako/Arduino/libra
Invalid version 'Internal RTC2.0' for library in
Invalid library found in /home/bako/Arduino/libra
Invalid library found in /home/bako/Arduino/libra
```

Figure 1.4 Selection of **WiFi forwarding port** and IP address

Select the port and click the **Download** button. In a few seconds, the new program will be downloaded. And you should see the **built-in LED flashing**.

To check the process, you can modify the program by modifying (for example) the value of:

```
int long const = 5000;
```

and download it again via WiFi to see the result.

To do :

1. Test the above programs.
2. Modify the code by adding a display on the OLED screen
3. Modify the code by adding sensor reading and display on OLED screen.

1.3 OTA on ESP32 card with WEB server

As in the previous solution, the first step is to download the code containing the OTA routine via USB. This is a mandatory step to be able to perform future updates/downloads via WiFi.

The new OTA program creates a **web server in STA mode**, accessible through a web browser. Once you are connected to the web server, you can download new programs with the OTA routine.
You can now upload new programs to the ESP32 by **generating and uploading the compiled** `.bin` file to the Arduino environment, via a web server.

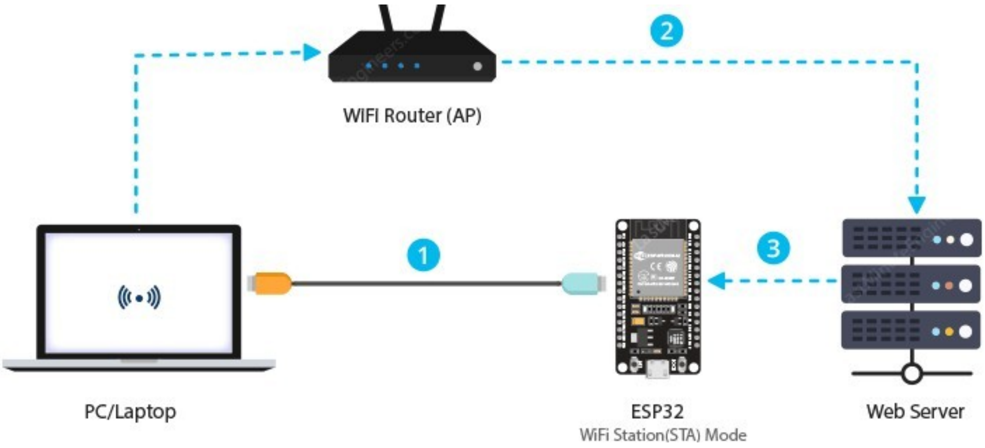


Figure 1.5 Transfer of the binary code (`.bin` file) by the WEB server to the ESP32 card

The ESP32 add-on for the Arduino IDE comes with an OTA library and an `OTAWebUpdater` example. You can access it via `File> Examples> ArduinoOTA> OTAWebUpdater` or via `github.`

To get started, connect your ESP32 to your computer and download the code below.
As usual you should offer the WiFi identifiers of your access point. so that ESP32 can establish a connection with the existing network.

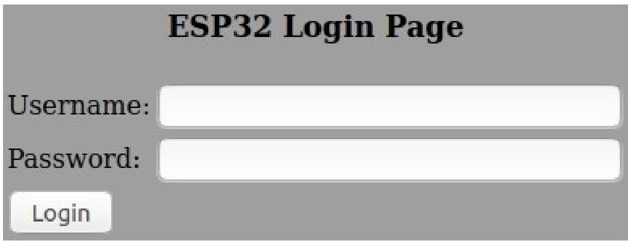
1.3.1 The starting program with Webserver

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

WebServer server(80);

// Login page
const char* loginIndex =
 "<form name='loginForm'>"
    "<table width='20%' bgcolor='A09F9F' align='center'>"
        "<tr>"
            "<td colspan=2>"
                "<center><font size=4><b>ESP32 Login Page</b></font></center>"
                "<br>"
            "</td>"
            "<br>"
            "<br>"
        "</tr>"
        "<td>Username:</td>"
        "<td><input type='text' size=25 name='userid'><br></td>"
        "</tr>"
        "<br>"
        "<br>"
        "<tr>"
            "<td>Password:</td>"
            "<td><input type='Password' size=25 name='pwd'><br></td>"
            "<br>"
            "<br>"
        "</tr>"
        "<tr>"
            "<td><input type='submit' onclick='check(this.form)' value='Login'></td>"
        "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
    "if(form.userid.value=='admin' && form.pwd.value=='admin')"
    "{"
    "window.open('/serverIndex')"
    "}"
    "else"
    "{"
    " alert('Error Password or Username')/*displays error message*/"
    "}"
    "}"
"</script>";

//Server Index Page
const char* serverIndex =
"<script src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
   "<input type='file' name='update'>"
        "<input type='submit' value='Update'>"
    "</form>"
 "<div id='prg'>progress: 0%</div>"
 "<script>"
  "$('form').submit(function(e){"
  "e.preventDefault();"
  "var form = $('#upload_form')[0];"
  "var data = new FormData(form);"
  " $.ajax({"
  "url: '/update',"
  "type: 'POST',"
  "data: data,"
  "contentType: false,"
  "processData:false,"
  "xhr: function() {"
  "var xhr = new window.XMLHttpRequest();"
  "xhr.upload.addEventListener('progress', function(evt) {"
  "if (evt.lengthComputable) {"
  "var per = evt.loaded / evt.total;"
  "$('#prg').html('progress: ' + Math.round(per*100) + '%');"
  "}"
  "}, false);"
  "return xhr;"
  "},"
  "success:function(d, s) {"
  "console.log('success!')"
 "},"
 "error: function (a, b, c) {"
 "}"
 "});"
 "});"
 "</script>";

void setup(void) {
  Serial.begin(9600);
  // Connect to WiFi network
  WiFi.begin(ssid, password);
  Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {
      delay(1000);
    }
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
}
```

1.3.2 Access the WEB server

The **OTAWebUpdater** program creates a web server in STA mode accessible via a web browser and which allows new programs to be downloaded to your ESP32 via WiFi.

To access the web server, open the serial monitor at a baud rate of **115200**. If all is well, the dynamic IP address obtained from your router (card) will be displayed.
Next, load a browser and point it to the IP address shown on the serial monitor. The ESP32 should serve as a web page requesting login information (default: **admin** and **admin**).



ESP32 Login Page

Figure 1.6 Initial **WEB server page** on the ESP32 board

If you want to change the user ID and password, edit the code below in your program.

`"if (form.userid.value == 'admin' && form.pwd.value == 'admin')"`

Once connected to the server, you will be redirected to the **`/serverIndex`** page.

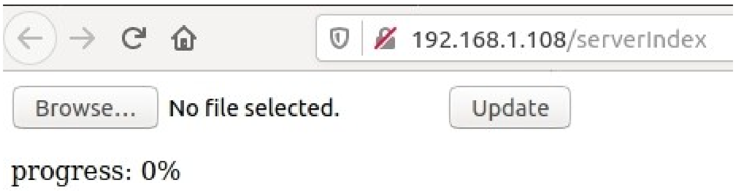


Figure 1.7 Page for finding and loading a **`.bin`** file

This page allows you to upload new programs to your ESP32 via WiFi. This new program being downloaded must be in binary **`.bin`** format.

1.3.3 Download a new program via WiFi (`.bin`)

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>
const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";
//variabls for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED
WebServer server(80);
/* Style */
String style =
"<style>#file-input,input{width:100%;height:44px;border-radius:4px;margin:10px auto;font-size:15px}"
"input{background:#f1f1f1;border:0;padding:0 15px}body{background:#3498db;font-family:sans-
serif;font-size:14px;color:#777}"
"#file-input{padding:0;border:1px solid #ddd;line-height:44px;text-
align:left;display:block;cursor:pointer}"
"#bar,#prgbar{background-color:#f1f1f1;border-radius:10px}#bar{background-
color:#3498db;width:0%;height:10px}"
"form{background:#fff;max-width:258px;margin:75px auto;padding:30px;border-radius:5px;text-
align:center}"
".btn{background:#3498db;color:#fff;cursor:pointer}</style>";
/* Login page */
String loginIndex =
"<form name=loginForm>"
"<h1>ESP32 Login</h1>"
"<input name=userid placeholder='User ID'> "
"<input name=pwd placeholder=Password type=Password> "
"<input type=submit onclick=check(this.form) class=btn value=Login></form>"
"<script>"
"function check(form) {"
"if(form.userid.value=='admin' && form.pwd.value=='admin')"
"{window.open('/serverIndex')}"
"else"
"{alert('Error Password or Username')}"
"}"
"</script>" + style;

String serverIndex =
"<script src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
"<input type='file' name='update' id='file' onchange='sub(this)' style=display:none>"
"<label id='file-input' for='file'>   Choose file...</label>"
"<input type='submit' class=btn value='Update'>"
"<br><br>"
"<div id='prg'></div>"
```

```
"<br><div id='prgbar'><div id='bar'></div></div><br></form>"
"<script>"
"function sub(obj){"
"var fileName = obj.value.split('\\\\');"
"document.getElementById('file-input').innerHTML = '   '+ fileName[fileName.length-1];"
"};"
"$('form').submit(function(e){"
"e.preventDefault();"
"var form = $('#upload_form')[0];"
"var data = new FormData(form);"
"$.ajax({"
"url: '/update',"
"type: 'POST',"
"data: data,"
"contentType: false,"
"processData:false,"
"xhr: function() {"
"var xhr = new window.XMLHttpRequest();"
"xhr.upload.addEventListener('progress', function(evt) {"
"if (evt.lengthComputable) {"
"var per = evt.loaded / evt.total;"
"$('#prg').html('progress: ' + Math.round(per*100) + '%');"
"$('#bar').css('width',Math.round(per*100) + '%');"
"}"
"}, false);"
"return xhr;"
"},"
"success:function(d, s) {"
"console.log('success!') "
"},"
"error: function (a, b, c) {"
"}"
"});"
"});"
"</script>" + style;

void setup(void) {
  Serial.begin(9600);pinMode(led,OUTPUT);
  WiFi.begin(ssid, password);Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  }
  Serial.println("");Serial.print("Connected to ");
  Serial.println(ssid);Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {  delay(1000);}
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
 unsigned long currentMillis = millis();
 if (currentMillis - previousMillis >= interval) {
 previousMillis = currentMillis;
 ledState = not(ledState);
 digitalWrite(led,  ledState);
 }
}
```

1.3.4 Generate a `.bin` file in the Arduino IDE

In order to upload a new sketch to the ESP32, we first need to **generate the compiled binary** **`.bin`** of your program. To do this, select **`Sketch > Export compiled Binary`**

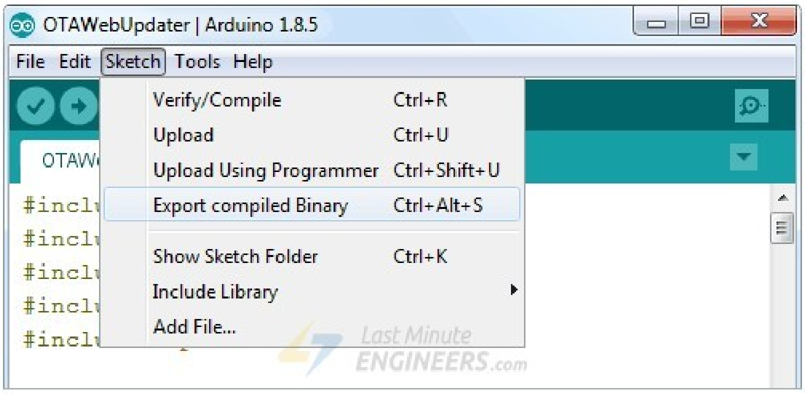
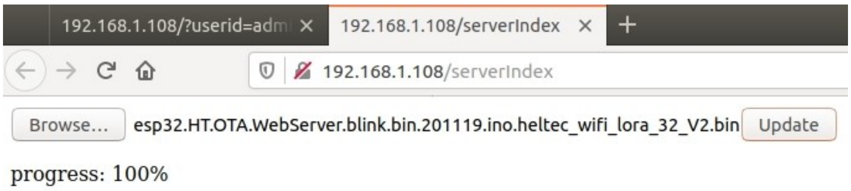

OTAWebUpdater | Arduino 1.8.5
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
OTAW
#incl
#incl
#incl
#incl
#incl
Last Minute ENGINEERS.com

Figure 1.8 Generation of binary code for transfer to the ESP32 board

1.3.5 Download a new sketch live on the ESP32

Once the **`.bin`** file is generated, you are now ready to upload the new code to the ESP32 via WiFi. Open the **`/serverIndex`** page in your browser. Click **`Choose File…`** Select the generated **`.bin`** file (in the same directory as your **`.ino`** sketch), then click **`Update`**.


192.168.1.108/?userid=adm
192.168.1.108/serverIndex
192.168.1.108/serverIndex
Browse...
esp32.HT.OTA.WebServer.blink.bin.201119.ino.heltec_wifi_lora_32_V2.bin
Update

progress: 100%

Figure 1.9 Transferring the binary code to the ESP32 board

In a few seconds, the new program will be downloaded. And you should see the built-in LED flashing.

To do

1. Test the above programs.
2. Modify the code by adding a display on the OLED screen
3. Modify the code by adding sensor reading and display on OLED screen.

1.4 Implementing OTA with the `WebOTA` Library

The previous solution requires the insertion of the HTML code of the WEB page created for the transfer of the **`.bin`** code to the ESP32 board. The **`WebOTA`** library promises to simplify this preparation.

You will need an inclusion, of course. If you don't mind using port 8080 and the default path **`/webota`**, just call handle**`_webota`**) from your **main `loop()`**

If you want to change the defaults, you need to add an extra call in your configuration. You also need to configure a few global variables to specify your network settings.

The only downside is that declarations with a **long `delay()`** in your loop can prevent some things from working correctly and are not a good idea. If you have any, you can replace all your **`delay()`** calls with **`webota_delay()`**, which will prevent the system from ignoring update requests.

The initial code must be loaded by the Arduino environment. To perform an update, simply access the ESP32 board with a web browser and use the correct port number and path. From here you can upload a new binary image created in the Arduino IDE.

Attention :
The code to load is not authenticated. This means anyone can upload code to your ESP. This may be acceptable on a private network, but on the Internet it is certainly problematic.

1.4.1 Initial code

```
#include <WebOTA.h>
#define LED_PIN 25
const char* host     = "ESP-OTA"; // Used for MDNS resolution
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  init_wifi(ssid, password, host);
  // Defaults to 8080 and "/webota" , webota.init(80, "/update");
}

void loop() {
  int md = 5000;
  digitalWrite(LED_PIN, HIGH);
  webota.delay(md);
  digitalWrite(LED_PIN, LOW);
  webota.delay(md);
  webota.handle();
}
```

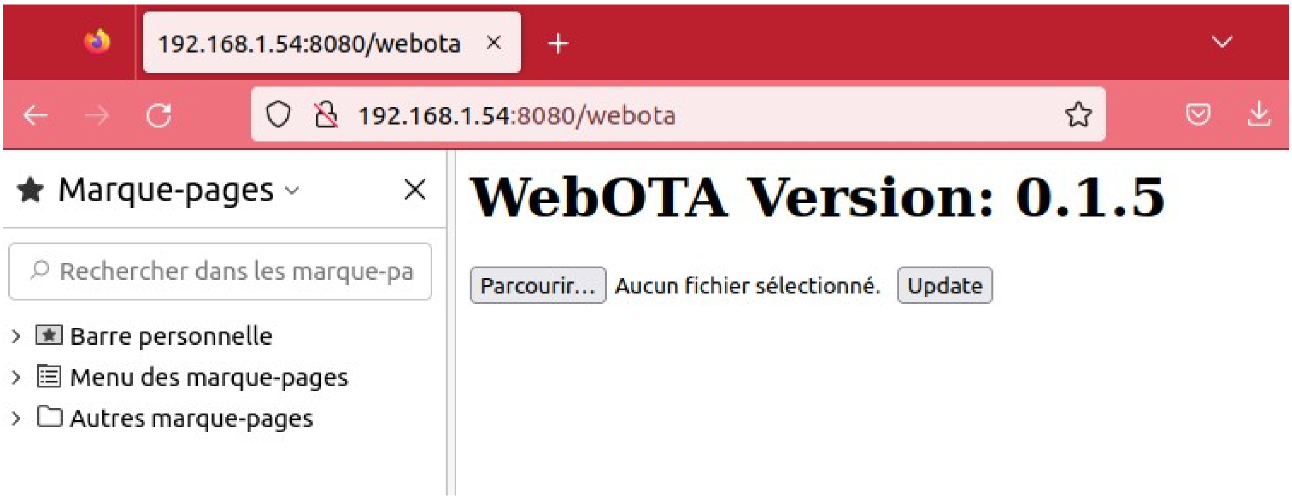
The terminal output:

```
Connecting to Wifi.
Connected to 'Livebox-08B0'

IP address   : 192.168.1.54
MAC address  : 7C:9E:BD:46:3A:7C
mDNS started : ESPOTA.local
WebOTA url   : http://ESPOTA.local:8080/webota
```

As you see above in the initial loading phase, we obtain the IP address of the WEB server on the ESP32.

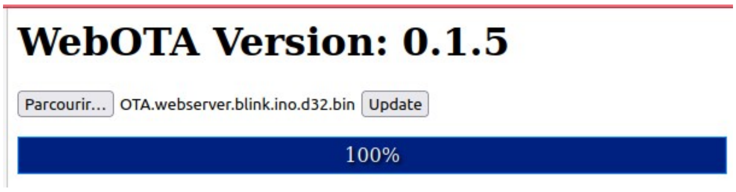
The display is up to the line with the `WebOTA` url. The server can be accessed with;

```
192.168.1.54:8080/webota
```



WebOTA Version: 0.1.5

It asks you to provide the location of your compiled Arduino code in `.bin`.



WebOTA Version: 0.1.5

Figure1.11 The initial page for loading the binary code from the WEB server

To do :

1. Test the example above.
2. Modify the code by adding a display on the OLED screen
3. Modify the code by adding sensor reading and display on OLED screen.

Table of Contents