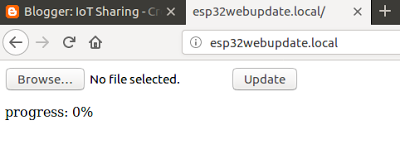
# **[Demo 36: Firmware update OTA via ESP Http Web Server](http://www.iotsharing.com/2017/11/firmware-update-ota-via-esp-http-webserver.html)**

**1. Introduction**  
In [Demo 34: firmware update OTA for ESP32 using HTTP and sdcard](http://www.iotsharing.com/2017/11/firmware-update-ota-for-esp32-using-http-sdcard.html) and [Demo 35: firmware update OTA for ESP32 directly using HTTP](http://www.iotsharing.com/2017/11/firmware-update-ota-for-esp32-using-http.html), I showed ways to update firmware OTA. In this demo, I will show you another way. That is updating firmware OTA for ESP via ESP Http Web server. With this demo, ESP will act as a web server and user will access the web server and upload the firmware file to ESP via web browser.

[](https://4.bp.blogspot.com/-VqeyOEv438M/WhQitKb5A_I/AAAAAAAAFs8/6LhPYW9uk6MQNOab47wil49GjO9vXDvIwCLcBGAs/s1600/esp32wota.png)

**Figure: Web interface of the demo**

User choose **Browse** button, navigate to firmware file and press **Update** button. The updating progress will be shown.

We will create a simple LED blink application, export the binary file for updating. In order to export the .bin file from Arduino IDE Menu, we choose **Sketch -> Export compiled Binary**.After finishing we choose **Sketch -> Show Sketch Folder**. You will se the .bin file there, rename it as **led.bin**

**2. Hardware**

Using [Demo 1](http://www.iotsharing.com/2017/05/blinky-hello-world-on-arduino-esp32.html" \t "http://www.iotsharing.com/2017/11/_blank)to connect ESP to LED.

**3. Software**

We will re-use the Web Server library in [Demo 12: How to turn the Arduino ESP32 into a Web Server.](http://www.iotsharing.com/2017/05/how-to-turn-esp32-into-web-server.html" \t "http://www.iotsharing.com/2017/11/_blank)Beside that, i also used the jquery library to create uploading Http POST request. MDNS ([Demo 9: How to use mDNS to resolve host names to Arduino ESP32 IP addresses](http://www.iotsharing.com/2017/05/how-to-use-mdns-to-resolve-hostname-esp32-ipaddress.html" \t "http://www.iotsharing.com/2017/11/_blank)) was used to resolve host name for our web server instead of using IP address directly. The code will be explained below.

|  |
| --- |
| #include <WiFi.h>  #include <WiFiClient.h>  #include <ESP32WebServer.h>  #include <ESPmDNS.h>  #include <Update.h>  const char\* host = "esp32webupdate";  const char\* ssid = "dd-wrt";  const char\* password = "0000000000";  ESP32WebServer server(80);  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(115200);  // Connect to WiFi network  WiFi.begin(ssid, password);  Serial.println("");  // Wait for connection  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)) {  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", 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);  } |

In the code, the variable **serverIndex**holds the **index** page which is return to the web browser firstly. We will use "**$.ajax**" to create asynchronous uploading request. This request will be handled by "/update" action at web server. We also use "**var xhr = new window.XMLHttpRequest()**" to handle the progress of uploading.   
The code "**MDNS.begin(host)**" will use MDNS to resolve "**http://esp32webupdate.local**" to our web server IP address.  
The code "**server.on("/", HTTP\_GET, []()**" will handle the first HTTP GET request from web browser and return the http status code 200 and the web page content in **serverIndex** variable.  
The code "**server.on("/update", HTTP\_POST, []()**" will handle the uploading firmware file process via HTTP POST. We handle the order of process via "**upload.status**" and use **Update**for flashing firmware. After finishing, we call "ESP.restart();" to restart ESP to get effect.