

ESP32 Useful Wi-Fi Library Functions (Arduino IDE)

This article is a compilation of useful Wi-Fi functions for the ESP32. We'll cover the following topics: scan Wi-Fi networks, connect to a Wi-Fi network, get Wi-Fi connection strength, check connection status, reconnect to the network after a connection is lost, Wi-Fi status, Wi-Fi modes, get the ESP32 IP address, set a fixed IP address and more.

This is not a novelty. There are plenty of examples of how to handle Wi-Fi with the ESP32. However, we thought it would be useful to compile some of the most used and practical Wi-Fi functions for the ESP32.



Table of Contents

Here's a list of what will be covered in this tutorial (you can click on the links to go to the corresponding section):

- [Wi-Fi Modes](#): station, access point and both (station + access point);
- [Scan Wi-Fi networks](#);
- [Connect to a Wi-Fi network](#);
- [Get Wi-Fi connection status](#);
- [Check Wi-Fi connection strength](#);
- [Get ESP32 IP address](#);
- [Set an ESP32 Static IP address](#);
- [Disconnect Wi-Fi](#);
- [Reconnect to Wi-Fi after connection is lost](#);
- [ESP32 Wi-Fi Events](#);
- [Reconnect to Wi-Fi Network After Lost Connection \(Wi-Fi Events\)](#);
- [ESP32 WiFiMulti](#)
- [Change ESP32 Hostname](#).

Including the Wi-Fi Library

The first thing you need to do to use the ESP32 Wi-Fi functionalities is to include the `wifi.h` library in your code, as follows:

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

- [Installing the ESP32 Board in Arduino IDE \(Windows, Mac OS X, Linux\)](#)

If you prefer to use [VS Code + PlatformIO](#), you just need to start a new project with an ESP32 board to be able to use the `WiFi.h` library and its functions.

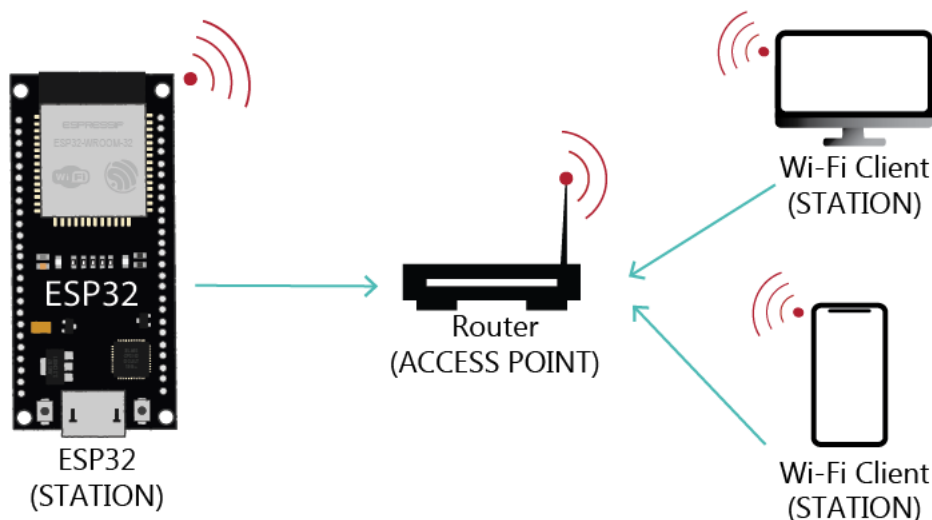
ESP32 Wi-Fi Modes

The ESP32 board can act as Wi-Fi Station, Access Point or both. To set the Wi-Fi mode, use `WiFi.mode()` and set the desired mode as argument:

<code>WiFi.mode(WIFI_STA)</code>	station mode: the ESP32 connects to an access point
<code>WiFi.mode(WIFI_AP)</code>	access point mode: stations can connect to the ESP32
<code>WiFi.mode(WIFI_AP_STA)</code>	access point and a station connected to another access point

Wi-Fi Station

When the ESP32 is set as a Wi-Fi station, it can connect to other networks (like your router). In this scenario, the router assigns a unique IP address to your ESP board. You can communicate with the ESP using other devices (stations) that are also connected to the same network by referring to the ESP unique IP address.



The router is connected to the internet, so we can request information from the internet using the ESP32 board like data from APIs (weather data, for example), publish data to online platforms, use icons and images from the internet or include JavaScript libraries to build web server pages.

Set the ESP32 as a Station and Connect to Wi-Fi Network

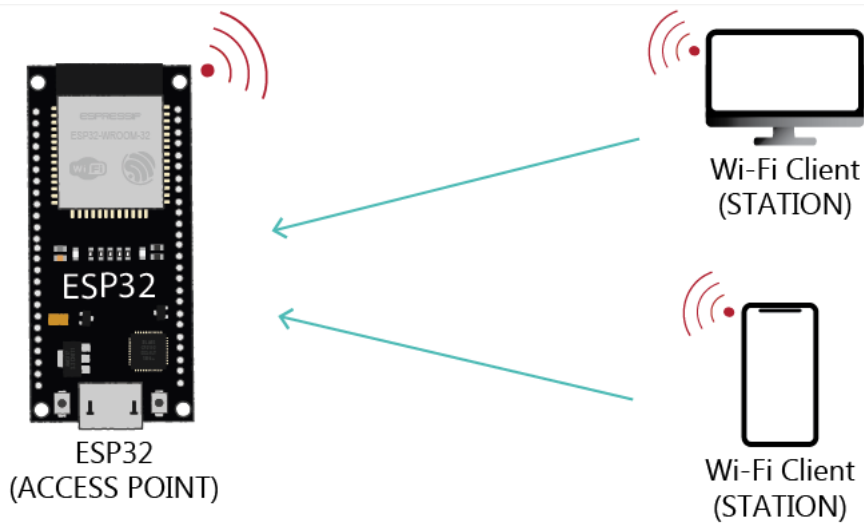
Go to [“Connect to Wi-Fi Network”](#) to learn how to set the ESP32 as station and connect it to a network.

In some cases, this might not be the best configuration – when you don’t have a network nearby and want you still want to connect to the ESP to control it. In this scenario, you must set your ESP board as an access point.

Access Point

When you set your ESP32 board as an access point, you can be connected using any device with Wi-Fi capabilities without connecting to your router. When you set the ESP32 as an access point, you create its own Wi-Fi network, and nearby Wi-Fi devices (stations) can connect to it, like your smartphone or computer. So, you don’t need to be connected to a router to control it.

This can be also useful if you want to have several ESP32 devices talking to each other without the need for a router



Because the ESP32 doesn't connect further to a wired network like your router, it is called soft-AP (soft Access Point). This means that if you try to load libraries or use firmware from the internet, it will not work. It also doesn't work if you make HTTP requests to services on the internet to publish sensor readings to the cloud or use services on the internet (like sending an email, for example).

Set the ESP32 as an Access Point

To set the ESP32 as an access point, set the Wi-Fi mode to access point:

```
WiFi.mode(WIFI_AP)
```

And then, use the `softAP()` method as follows:

```
WiFi.softAP(ssid, password);
```

`ssid` is the name you want to give to the ESP32 access point, and the `password` variable is the password for the access point. If you don't want to set a password, set it to `NULL`.

There are also other optional parameters you can pass to the `softAP()` method. Here are all the parameters:

```
WiFi.softAP(const char* ssid, const char* password, int channel, int ssid_hidden, int max_connection)
```

- `ssid`: name for the access point – maximum of 63 characters;
- `password`: **minimum of 8 characters**; set to `NULL` if you want the access point to be open;
- `channel`: Wi-Fi channel number (1-13)
- `ssid_hidden`: (0 = broadcast SSID, 1 = hide SSID)
- `max_connection`: maximum simultaneous connected clients (1-4)

We have a complete tutorial explaining how to set up the ESP32 as an access point:

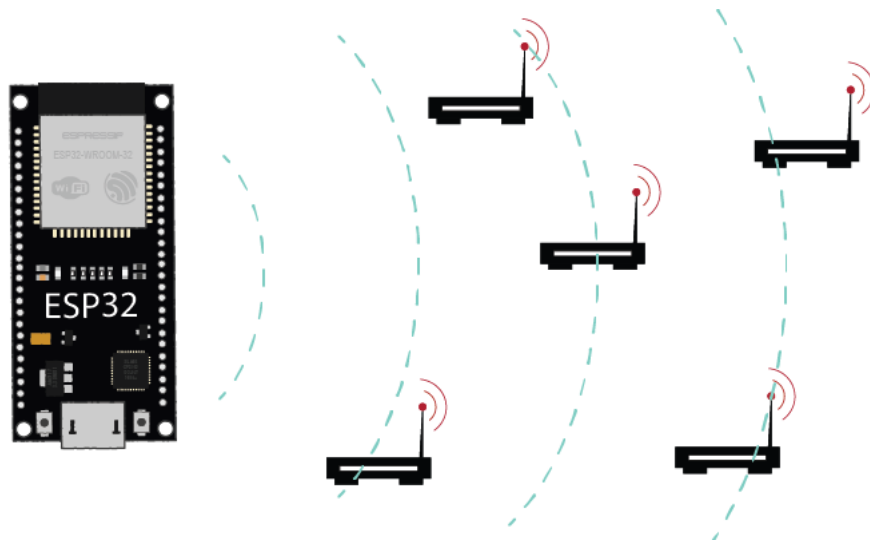
- [How to Set an ESP32 Access Point \(AP\) for Web Server](#)

Wi-Fi Station + Access Point

The ESP32 can be set as a Wi-Fi station and access point simultaneously. Set its mode to `WIFI_AP_STA`.

Scan Wi-Fi Networks

The ESP32 can scan nearby Wi-Fi networks within its Wi-Fi range. In your Arduino IDE, go to **File > Examples > WiFi > WiFiScan**. This will load a sketch that scans Wi-Fi networks within the range of your ESP32 board.



This can be useful to check if the Wi-Fi network you're trying to connect is within the range of your board or other applications. Your Wi-Fi project may not often work because it may not be able to connect to your router due to insufficient Wi-Fi strength.

Here's the example:

```
#include "WiFi.h"

void setup() {
  Serial.begin(115200);

  // Set WiFi to station mode and disconnect from an AP if it was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");
}

void loop() {
  Serial.println("scan start");

  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();
  Serial.println("scan done");
  if (n == 0) {
    Serial.println("no networks found");
  } else {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
```

[View raw code](#)

`WiFi.scanNetworks()` returns the number of networks found.

```
int n = WiFi.scanNetworks();
```

After the scanning, you can access the parameters about each network.

`WiFi.SSID()` prints the SSID for a specific network:

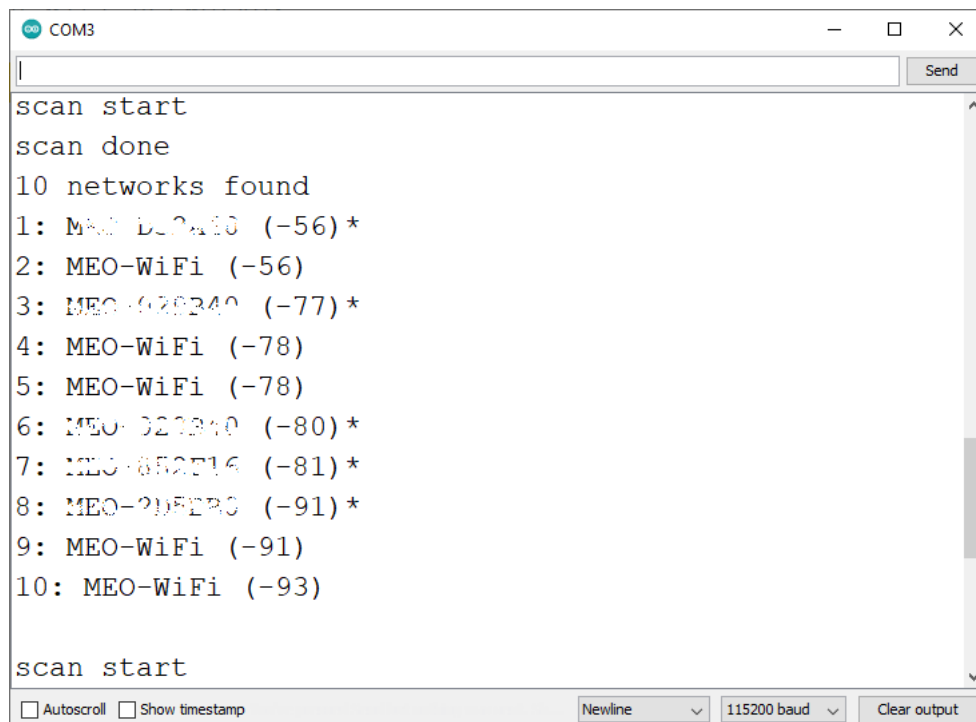
```
Serial.print(WiFi.SSID(i));
```

`WiFi.RSSI()` returns the RSSI of that network. RSSI stands for **R**eceived **S**ignal **S**trength **I**ndicator. It is an estimated measure of power level that an RF client device is receiving from an access point or router.

```
Serial.print(WiFi.RSSI(i));
```

Finally, `WiFi.encryptionType()` returns the network encryption type. That specific example puts a * in the case of open networks. However, that function can return one of the following options (not just open networks):

- `WIFI_AUTH_OPEN`
- `WIFI_AUTH_WEP`
- `WIFI_AUTH_WPA_PSK`
- `WIFI_AUTH_WPA2_PSK`
- `WIFI_AUTH_WPA_WPA2_PSK`
- `WIFI_AUTH_WPA2_ENTERPRISE`



```
COM3
scan start
scan done
10 networks found
1: MAC: DC2440 (-56) *
2: MEO-WiFi (-56)
3: MEO-020B40 (-77) *
4: MEO-WiFi (-78)
5: MEO-WiFi (-78)
6: MEO-020B40 (-80) *
7: MEO-050E16 (-81) *
8: MEO-200F00 (-91) *
9: MEO-WiFi (-91)
10: MEO-WiFi (-93)

scan start
```

Connect to a Wi-Fi Network

To connect the ESP32 to a specific Wi-Fi network, you must know its SSID and password. Additionally, that network must be within the ESP32 Wi-Fi range (to check that, you can use the previous example to scan Wi-Fi networks).

You can use the following function to connect the ESP32 to a Wi-Fi network `initWiFi()` :

```
void initWiFi() {  
    WiFi.mode(WIFI_STA);  
    WiFi.begin(ssid, password);  
    Serial.print("Connecting to WiFi ..");  
    while (WiFi.status() != WL_CONNECTED) {  
        Serial.print('.');  
        delay(1000);  
    }  
    Serial.println(WiFi.localIP());  
}
```

The `ssid` and `password` variables hold the SSID and password of the network you want to connect to.

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

Then, you simply need to call the `initWiFi()` function in your `setup()`.

How it Works?

Let's take a quick look on how this function works.

First, set the Wi-Fi mode. If the ESP32 will connected to another network (access point/hotspot) it must be in station mode.

```
WiFi.mode(WIFI_STA);
```

Then, use `WiFi.begin()` to connect to a network. You must pass as arguments the network SSID and its password:

```
WiFi.begin(ssid, password);
```

Connecting to a Wi-Fi network can take a while, so we usually add a while loop that keeps checking if the connection was already established by using `WiFi.status()`. When the connection is successfully established, it returns `WL_CONNECTED`.

```
while (WiFi.status() != WL_CONNECTED) {
```

Get Wi-Fi Connection Status

To get the status of the Wi-Fi connection, you can use `WiFi.status()`. This returns one of the following values that correspond to the constants on the table:

Value	Constant	Meaning
0	<code>WL_IDLE_STATUS</code>	temporary status assigned when <code>WiFi.begin()</code> is called
1	<code>WL_NO_SSID_AVAIL</code>	when no SSID are available
2	<code>WL_SCAN_COMPLETED</code>	scan networks is completed
3	<code>WL_CONNECTED</code>	when connected to a WiFi network

5	WL_CONNECTION_LOST	when the connection is lost
6	WL_DISCONNECTED	when disconnected from a network

Get WiFi Connection Strength

To get the WiFi connection strength, you can simply call `WiFi.RSSI()` after a WiFi connection.

Here's an example:

```
#include <WiFi.h>

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

void initWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}

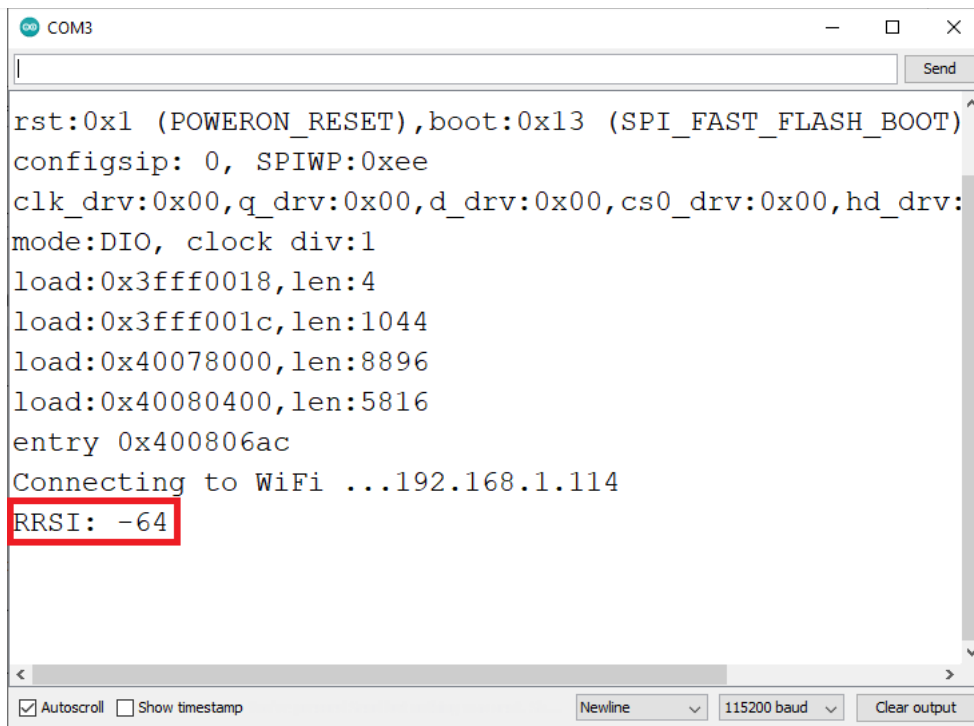
void setup() {
  Serial.begin(115200);
  initWiFi();
  Serial.print("RSSI: ");
  Serial.println(WiFi.RSSI());
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

[View raw code](#)

Insert your network credentials and upload the code.

Open the Serial Monitor and press the ESP32 on-board RST button. It will connect to your network and print the RSSI (received signal strength indicator).



```
COM3
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:8896
load:0x40080400,len:5816
entry 0x400806ac
Connecting to WiFi ...192.168.1.114
RRSI: -64
```

A lower absolute value means a strongest Wi-Fi connection.

Get ESP32 IP Address

When the ESP32 is set as a Wi-Fi station, it can connect to other networks (like your router). In this scenario, the router assigns a unique IP address to your ESP32 board. To get your board's IP address, you need to call `WiFi.localIP()` after establishing a connection with your network.

```
Serial.println(WiFi.localIP());
```

Set a Static ESP32 IP Address

Instead of getting a randomly assigned IP address, you can set an available IP address of your preference to the ESP32 using `WiFi.config()`.

Outside the `setup()` and `loop()` functions, define the following variables with your own static IP address and corresponding gateway IP address. By default, the following code assigns the IP address 192.168.1.184 that works in the gateway 192.168.1.1.

```
// Set your Static IP address
IPAddress local_IP(192, 168, 1, 184);
// Set your Gateway IP address
IPAddress gateway(192, 168, 1, 1);

IPAddress subnet(255, 255, 0, 0);
IPAddress primaryDNS(8, 8, 8, 8); // optional
IPAddress secondaryDNS(8, 8, 4, 4); // optional
```

Then, in the `setup()` you need to call the `WiFi.config()` method to assign the configurations to your ESP32.

```
// Configures static IP address
if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
```



```
Serial.println("STA Failed to configure");
}
```

The `primaryDNS` and `secondaryDNS` parameters are optional and you can remove them.

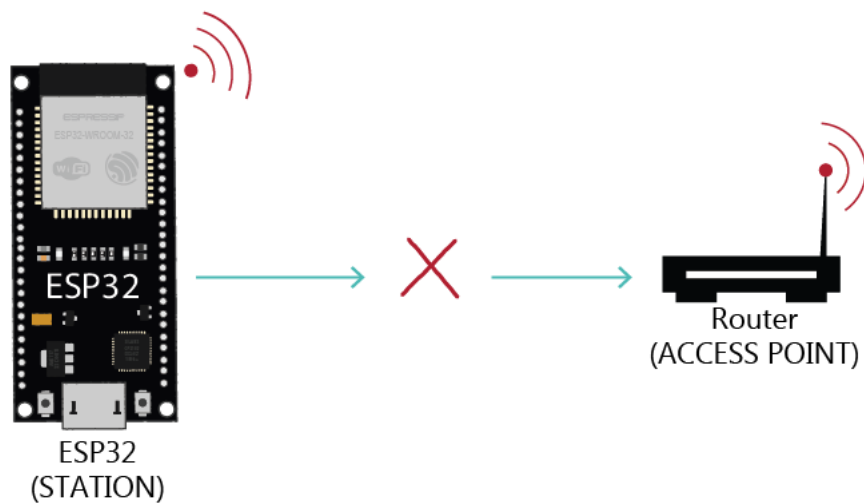
We recommend reading the following tutorial to learn how to set a static IP address:

- [ESP32 Static/Fixed IP Address](#)

Disconnect from Wi-Fi Network

To disconnect from a previously connected Wi-Fi network, use `WiFi.disconnect()` :

```
WiFi.disconnect()
```



Reconnect to Wi-Fi Network After Lost Connection

To reconnect to Wi-Fi after a connection is lost, you can use `WiFi.reconnect()` to try to reconnect to the previously connected access point:

```
WiFi.reconnect()
```

Or, you can call `WiFi.disconnect()` followed by `WiFi.begin(ssid,password)` .

```
WiFi.disconnect();
WiFi.begin(ssid, password);
```

Alternatively, you can also try to restart the ESP32 with `ESP.restart()` when the connection is lost.

You can add something like the snippet below to your `loop()` that checks once in a while if the board is connected.

```
unsigned long currentMillis = millis();
// if WiFi is down, try reconnecting
if ((WiFi.status() != WL_CONNECTED) && (currentMillis - previousMillis >=interval)) {
    Serial.print(millis());
    Serial.println("Reconnecting to WiFi...");
}
```

```
    previousMillis = currentMillis;
}
```

Don't forget to declare the `previousMillis` and `interval` variables. The `interval` corresponds to the period of time between each check in milliseconds (for example 30 seconds):

```
unsigned long previousMillis = 0;
unsigned long interval = 30000;
```

Here's a complete example.

```
#include <WiFi.h>

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

unsigned long previousMillis = 0;
unsigned long interval = 30000;

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi ..");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print('.');
        delay(1000);
    }
    Serial.println(WiFi.localIP());
}

void setup() {
    Serial.begin(115200);
    initWiFi();
    Serial.print("RSSI: ");
    Serial.println(WiFi.RSSI());
}

void loop() {
```

[View raw code](#)

This example shows how to connect to a network and checks every 30 seconds if it is still connected. If it isn't, it disconnects and tries to reconnect again.

You can read our guide: [\[SOLVED\] Reconnect ESP32 to Wi-Fi Network After Lost Connection](#).

Alternatively, you can also use WiFi Events to detect that the connection was lost and call a function to handle what to do when that happens (see the next section).

ESP32 Wi-Fi Events

The ESP32 can handle all the following Wi-Fi events (check the [source code](#)):

1	ARDUINO_EVENT_WIFI_SCAN_DONE	ESP32 finishes scanning AP
2	ARDUINO_EVENT_WIFI_STA_START	ESP32 station start
3	ARDUINO_EVENT_WIFI_STA_STOP	ESP32 station stop
4	ARDUINO_EVENT_WIFI_STA_CONNECTED	ESP32 station connected to AP
5	ARDUINO_EVENT_WIFI_STA_DISCONNECTED	ESP32 station disconnected from AP
6	ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE	the auth mode of AP connected by ESP32 station changed
7	ARDUINO_EVENT_WIFI_STA_GOT_IP	ESP32 station got IP from connected AP
8	ARDUINO_EVENT_WIFI_STA_LOST_IP	ESP32 station lost IP and the IP is reset to 0
9	ARDUINO_EVENT_WPS_ER_SUCCESS	ESP32 station wps succeeds in enrollee mode
10	ARDUINO_EVENT_WPS_ER_FAILED	ESP32 station wps fails in enrollee mode
11	ARDUINO_EVENT_WPS_ER_TIMEOUT	ESP32 station wps timeout in enrollee mode
12	ARDUINO_EVENT_WPS_ER_PIN	ESP32 station wps pin code in enrollee mode
13	ARDUINO_EVENT_WIFI_AP_START	ESP32 soft-AP start
14	ARDUINO_EVENT_WIFI_AP_STOP	ESP32 soft-AP stop
15	ARDUINO_EVENT_WIFI_AP_STACONNECTED	a station connected to ESP32 soft-AP
16	ARDUINO_EVENT_WIFI_AP_STADISCONNECTED	a station disconnected from ESP32 soft-AP
17	ARDUINO_EVENT_WIFI_AP_STAIPASSIGNED	ESP32 soft-AP assign an IP to a connected station
18	ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED	Receive probe request packet in soft-AP interface
19	ARDUINO_EVENT_WIFI_AP_GOT_IP6	ESP32 access point v6IP addr is preferred
19	ARDUINO_EVENT_WIFI_STA_GOT_IP6	ESP32 station v6IP addr is preferred
19	ARDUINO_EVENT_ETH_GOT_IP6	Ethernet IPv6 is preferred
20	ARDUINO_EVENT_ETH_START	ESP32 ethernet start
21	ARDUINO_EVENT_ETH_STOP	ESP32 ethernet stop
22	ARDUINO_EVENT_ETH_CONNECTED	ESP32 ethernet phy link up
23	ARDUINO_EVENT_ETH_DISCONNECTED	ESP32 ethernet phy link down
24	ARDUINO_EVENT_ETH_GOT_IP	ESP32 ethernet got IP from connected AP
25	ARDUINO_EVENT_MAX	

For a complete example on how to use those events, in your Arduino IDE, go to **File > Examples > WiFi > WiFiClientEvents**.

```

/* This sketch shows the WiFi event usage - Example from WiFi > WiFiClientEvents
   Complete details at https://RandomNerdTutorials.com/esp32-useful-wi-fi-functions-arduino/
*/
* WiFi Events

```

1	ARDUINO_EVENT_WIFI_SCAN_DONE	< ESP32 finish scanning AP
2	ARDUINO_EVENT_WIFI_STA_START	< ESP32 station start
3	ARDUINO_EVENT_WIFI_STA_STOP	< ESP32 station stop
4	ARDUINO_EVENT_WIFI_STA_CONNECTED	< ESP32 station connected to AP
5	ARDUINO_EVENT_WIFI_STA_DISCONNECTED	< ESP32 station disconnected from AP
6	ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE	< the auth mode of AP connected by ESP32 sta
7	ARDUINO_EVENT_WIFI_STA_GOT_IP	< ESP32 station got IP from connected AP
8	ARDUINO_EVENT_WIFI_STA_LOST_IP	< ESP32 station lost IP and the IP is reset
9	ARDUINO_EVENT_WPS_ER_SUCCESS	< ESP32 station wps succeeds in enrollee mode
10	ARDUINO_EVENT_WPS_ER_FAILED	< ESP32 station wps fails in enrollee mode
11	ARDUINO_EVENT_WPS_ER_TIMEOUT	< ESP32 station wps timeout in enrollee mode
12	ARDUINO_EVENT_WPS_ER_PIN	< ESP32 station wps pin code in enrollee mode
13	ARDUINO_EVENT_WIFI_AP_START	< ESP32 soft-AP start
14	ARDUINO_EVENT_WIFI_AP_STOP	< ESP32 soft-AP stop
15	ARDUINO_EVENT_WIFI_AP_STACONNECTED	< a station connected to ESP32 soft-AP
16	ARDUINO_EVENT_WIFI_AP_STADISCONNECTED	< a station disconnected from ESP32 soft-AP
17	ARDUINO_EVENT_WIFI_AP_STAIPASSIGNED	< ESP32 soft-AP assign an IP to a connected
18	ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED	< Receive probe request packet in soft-AP in
19	ARDUINO_EVENT_WIFI_AP_GOT_IP6	< ESP32 ap interface v6IP addr is preferred
19	ARDUINO_EVENT_WIFI_STA_GOT_IP6	< ESP32 station interface v6IP addr is preferred

[View raw code](#)

With Wi-Fi Events, you don't need to be constantly checking the Wi-Fi state. When a certain event happens, it automatically calls the corresponding handling function.

Reconnect to Wi-Fi Network After Lost Connection (Wi-Fi Events)

Wi-Fi events can be useful to detect that a connection was lost and try to reconnect right after (use the `SYSTEM_EVENT_AP_STADISCONNECTED` event). Here's a sample code:

```
#include <WiFi.h>

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

void WiFiStationConnected(WiFiEvent_t event, WiFiEventInfo_t info){
    Serial.println("Connected to AP successfully!");
}

void WiFiGotIP(WiFiEvent_t event, WiFiEventInfo_t info){
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}
```

How it Works?

In this example we've added three Wi-Fi events: when the ESP32 connects, when it gets an IP address, and when it disconnects: `ARDUINO_EVENT_WIFI_STA_CONNECTED`, `ARDUINO_EVENT_WIFI_STA_GOT_IP`, `ARDUINO_EVENT_WIFI_STA_DISCONNECTED`.

When the ESP32 station connects to the access point (`ARDUINO_EVENT_WIFI_STA_CONNECTED` event), the `WiFiStationConnected()` function will be called:

```
WiFi.onEvent(WiFiStationConnected, WiFiEvent_t::ARDUINO_EVENT_WIFI_STA_CONNECTED);
```

The `WiFiStationConnected()` function simply prints that the ESP32 connected to an access point (for example, your router) successfully. However, you can modify the function to do any other task (like light up an LED to indicate that it is successfully connected to the network).

```
void WiFiStationConnected(WiFiEvent_t event, WiFiEventInfo_t info){
    Serial.println("Connected to AP successfully!");
}
```

When the ESP32 gets its IP address, the `WiFiGotIP()` function runs.

```
WiFi.onEvent(WiFiGotIP, WiFiEvent_t::ARDUINO_EVENT_WIFI_STA_GOT_IP);
```

That function simply prints the IP address on the Serial Monitor.

```
void WiFiGotIP(WiFiEvent_t event, WiFiEventInfo_t info){
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}
```

When the ESP32 loses the connection with the access point (`ARDUINO_EVENT_WIFI_STA_DISCONNECTED`), the `WiFiStationDisconnected()` function is called.

```
WiFi.onEvent(WiFiStationDisconnected, WiFiEvent_t::ARDUINO_EVENT_WIFI_STA_DISCONNECTED);
```

That function prints a message indicating that the connection was lost and tries to reconnect:

```
void WiFiStationDisconnected(WiFiEvent_t event, WiFiEventInfo_t info){
    Serial.println("Disconnected from WiFi access point");
    Serial.print("WiFi lost connection. Reason: ");
    Serial.println(info.wifi_sta_disconnected.reason);
    Serial.println("Trying to Reconnect");
    WiFi.begin(ssid, password);
}
```

The ESP32 WiFiMulti allows you to register multiple networks (SSID/password combinations). The ESP32 will connect to the Wi-Fi network with the strongest signal (RSSI). If the connection is lost, it will connect to the next network on the list. This requires that you include the `WiFiMulti.h` library (you don't need to install it, it comes by default with the ESP32 package).

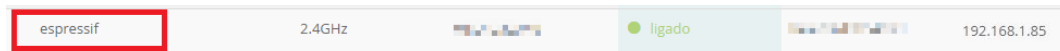
To learn how to use WiFiMulti, read the following tutorial:

- [ESP32 WiFiMulti: Connect to the Strongest Wi-Fi Network \(from a list of networks\)](#)

Change ESP32 Hostname

To set a custom hostname for your board, call `WiFi.setHostname(YOUR_NEW_HOSTNAME);` before `WiFi.begin();`

The default ESP32 hostname is `espressif`.



There is a method provided by the `WiFi.h` library that allows you to set a custom hostname.

First, start by defining your new hostname. For example:

```
String hostname = "ESP32 Node Temperature";
```

Then, call the `WiFi.setHostname()` function before calling `WiFi.begin()`. You also need to call `WiFi.config()` as shown below:

```
WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);  
WiFi.setHostname(hostname.c_str()); //define hostname
```

You can copy the complete example below:

```
#include <WiFi.h>  
  
// Replace with your network credentials (STATION)  
const char* ssid = "REPLACE_WITH_YOUR_SSID";  
const char* password = "REPLACE_WITH_YOUR_PASSWORD";  
  
String hostname = "ESP32 Node Temperature";  
  
void initWiFi() {  
  WiFi.mode(WIFI_STA);  
  WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);  
  WiFi.setHostname(hostname.c_str()); //define hostname  
  //wifi_station_set_hostname( hostname.c_str() );  
}
```

```
initWiFi();  
Serial.print("RSSI: ");
```

[View raw code](#)

You can use this previous snippet of code in your projects to set a custom hostname for the ESP32.

Important: you may need to restart your router for the changes to take effect.

After this, if you go to your router settings, you'll see the ESP32 with the custom hostname.



Wrapping Up

This article was a compilation of some of the most used and useful ESP32 Wi-Fi functions. Although there are plenty of examples of using the ESP32 Wi-Fi capabilities, there is little documentation explaining how to use the Wi-Fi functions with the ESP32 using Arduino IDE. So, we've decided to put together this guide to make it easier to use ESP32 Wi-Fi-related functions in your projects.

If you have other suggestions, you can share them in the comments section.

We hope you've found this tutorial useful.

Learn more about the ESP32 with our resources:

- [Learn ESP32 with Arduino IDE](#)
- [Build Web Servers with ESP32 and ESP8266](#)
- [More ESP32 Projects and Tutorials...](#)

**PCBWay**
PCB Fabrication & Assembly

ONLY \$5 for 10 PCBs

✓ 24-hour Build Time

✓ Quality Guaranteed

✓ Most Soldermask Colors:

■

■

■

■

■

■

■

■

■

■

[Order now](#)

www.pcbway.com



SMART HOME with Raspberry Pi, ESP32, ESP8266 [eBook]

Learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database [DOWNLOAD »](#)

Recommended Resources