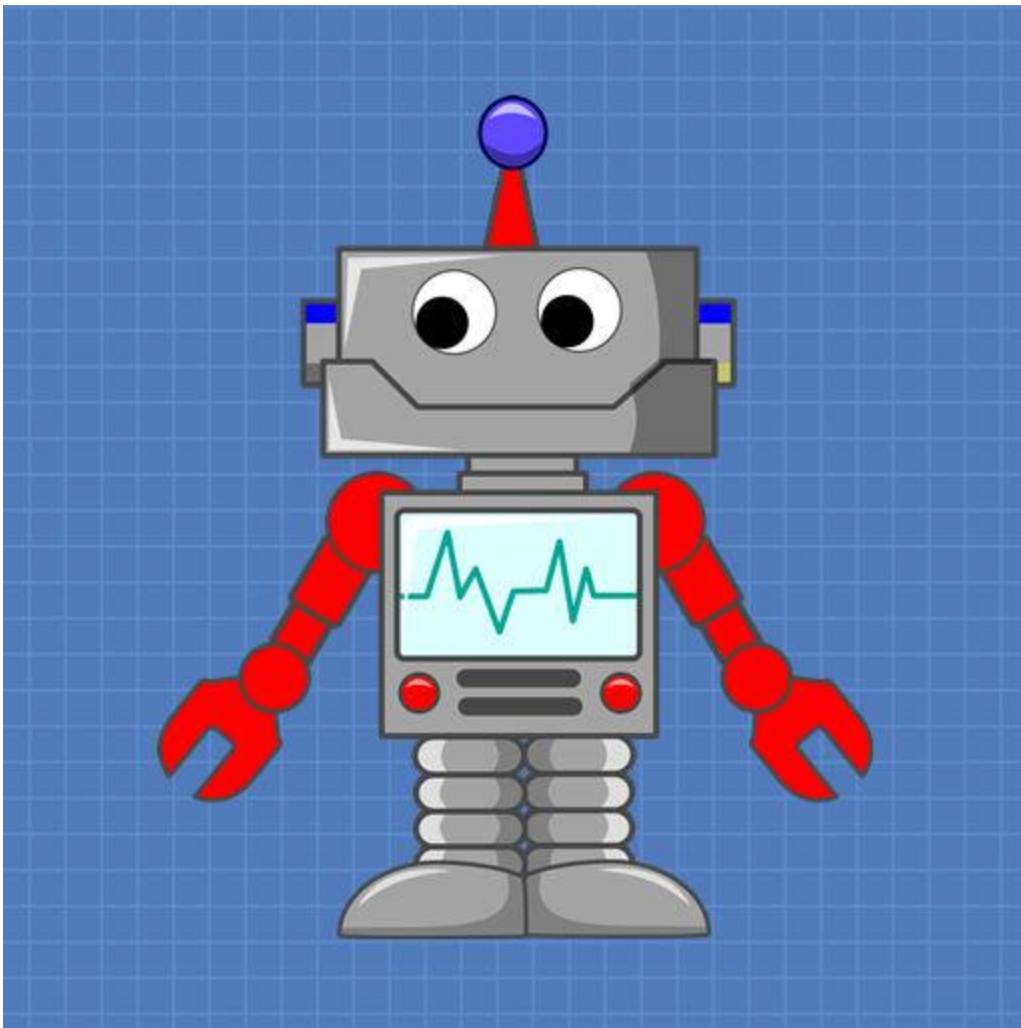


ESP32 WiFiManager – Easy WiFi Provisioning



DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Today we will see how to use WiFiManager to provide an SSID, password, and other configuration information to an ESP32. Stop hard-coding your WiFi information!



Introduction

The ESP32 is an amazing microcontroller. It's inexpensive yet powerful, boasting a 32-bit processor and many models have dual-cores. It has a wealth of I/O ports, several 12-bit A/D converters, a digital to analog converter, and I2C, I2S, SPI, and UART communications.

<https://dronebotworkshop.com>

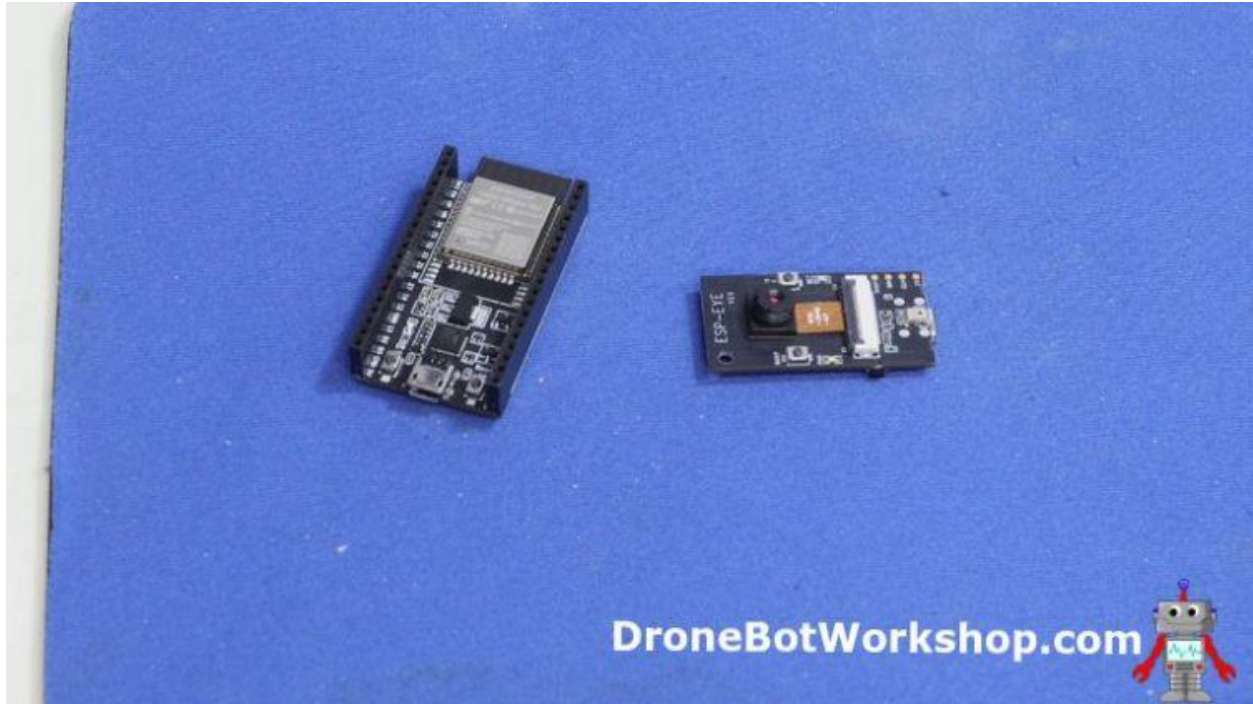
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



It even has touch switch capabilities, and many models have built-in cameras, displays, and/or microphones.

But despite all of those capabilities, the feature that most experimenters cherish in the ESP32 is its built-in WiFi and Bluetooth capabilities. This opens up a whole world of remote control and internet-enabled applications.

<https://dronebotworkshop.com>



Working with WiFi on the ESP32 is effortless, thanks to the included WiFi Library. But there is a weak point in the system, and that is getting connected to the WiFi network in the first place.

Connecting to WiFi

The normal method of connecting to a WiFi network in an ESP32 sketch is to code your network login parameters (SSID and password) directly in plain text. While this is certainly sufficient when you are just experimenting, it has a number of disadvantages:

- Your ESP32 can ONLY connect to the network you've hard-coded.
- Changing WiFi networks requires a code edit and recompilation.
- You need to edit that data out of your code if you share or distribute it.
- You can't create a commercial product using hard-coded WiFi parameters.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

A preferable method would be to use a system that lets you configure your WiFi credentials after the code has been compiled and loaded onto your ESP32. This type of system is known as “WiFi Provisioning”.

WiFi Provisioning

What exactly is “WiFi Provisioning”?

Provisioning is the process of preparing and equipping a network to allow it to provide new services. When we are dealing with WiFi it involves providing some credentials to connect to a network and obtain an IP address.

Connecting to a WiFi network requires two pieces of information:

- **The Network SSID** – SSID stands for *Service Set Identifier* and is your network’s name.
- **The Network Password** – Used for a secured WiFi network, which most are (and I hope yours is).

One method of provisioning a WiFi network is to use a keypad and display to enter these credentials. If your project already has these pieces of hardware, then this is a feasible option.

But many of our ESP32 projects, in fact, most of them, don’t have a keyboard or display. And it wouldn’t really be cost-effective to add them just to connect to the WiFi network.

What we need is a method of doing this without any additional hardware. And we can look at the world of commercial devices to get some great examples.

IoT Devices and Computer Peripherals

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Commercial IoT devices come in all shapes and sizes, from cameras and displays to temperature sensors and light bulbs. And they all require a connection to a WiFi network to function.

Obviously, putting a control panel on each of these devices to allow WiFi setup would be impossible, imagine how you'd accomplish this with a light bulb!

Then we have computer peripherals, specifically printers, and scanners. While many of these devices do have displays, they usually have a limited, if any, input device. It wouldn't be that easy to enter your WiFi credentials on a device with only four buttons on its keypad, especially if you use a secure password with a mix of numeric, alphabetical, and punctuation characters.

These devices employ a few different methods to achieve WiFi connectivity.

Provisioning Methods

There are several ways we can accomplish WiFi Provisioning. We are going to look at three of them today:

- **SmartConfig** – A method developed by Texas Instruments. It requires an app on an Android or IOS device to configure the WiFi credentials.
- **WPS** – WiFi Protected Setup, which uses a pushbutton on both the router and target device to achieve pairing and connectivity. Not every router supports WPS.
- **Local Access Point** – This sets up the target device initially as a web server at a fixed address. The server displays a configuration page and an external device (computer, tablet, phone) is used to configure it and then set it into WiFi slave mode.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Each of these methods has strengths and weaknesses, and note every one of them is suitable for every application.

The method we will be using today is the *Local Access Point* method.

This chart compares the features and requirements of the three provisioning methods:

WiFiManager

Local Access Point (or Local AP for short) is an ideal method to use with the ESP32. It allows you to use a phone, tablet, or WiFi-capable computer to set up your ESP32 WiFi credentials, as well as any additional configuration parameters you might require.

WiFiManager is a library that was originally written to perform this task using an ESP8266, and it has now been expanded to work with the ESP32.

WiFiManager works as follows:

- 1 – The ESP32 Boots up and checks to see if it already has a WiFi network configured.
- 2 – If it does have WiFi credentials, it uses them to log into a network. Assuming it is successful, then no other action is required, and it can start running its sketch.
- 3 – If it does not have WiFi credentials, or if they are invalid, it sets itself up as an Access Point instead. An Access Point is where the ESP32 provides its own WiFi network for other devices to connect to.
- 4 – It creates a webpage at an address of 192.168.4.1. On the webpage, there is a list of available WiFi networks that you can join, plus text boxes for entering your SSID and

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

password. You can also configure it to have additional text boxes (or other HTML elements) to grab other configuration information you might require.

5 – The user enters the appropriate information and presses the Save button. The information is transmitted back to the ESP32.

6 – The ESP32 changes back to Station mode and attempts to connect to the desired WiFi network using the supplied credentials.

7 – Assuming that the user has supplied the correct login credentials, the ESP32 will now be connected to the WiFi network.

This is not a system unique to WiFiManager, and there are several code samples that would allow you to implement this. The beauty of WiFiManager is that it is all done for you in the library. You don't need to code any HTML (unless you want to) to make it work. It's really just a few lines of code.

Basic WiFiManager Tests

The best way to learn about something, of course, is to put it to use. So let's get WiFiManager installed and start using it!

We will be using the Arduino IDE. I used the classic (version 1.8.xx) IDE, but you can also use the newer [Arduino IDE 2.0](#).

I am going to assume that you have already set up your Arduino IDE to use the ESP32, so I won't go over that step in this article. If you haven't done that, then please refer to my article on [getting started with the ESP32](#) for details.

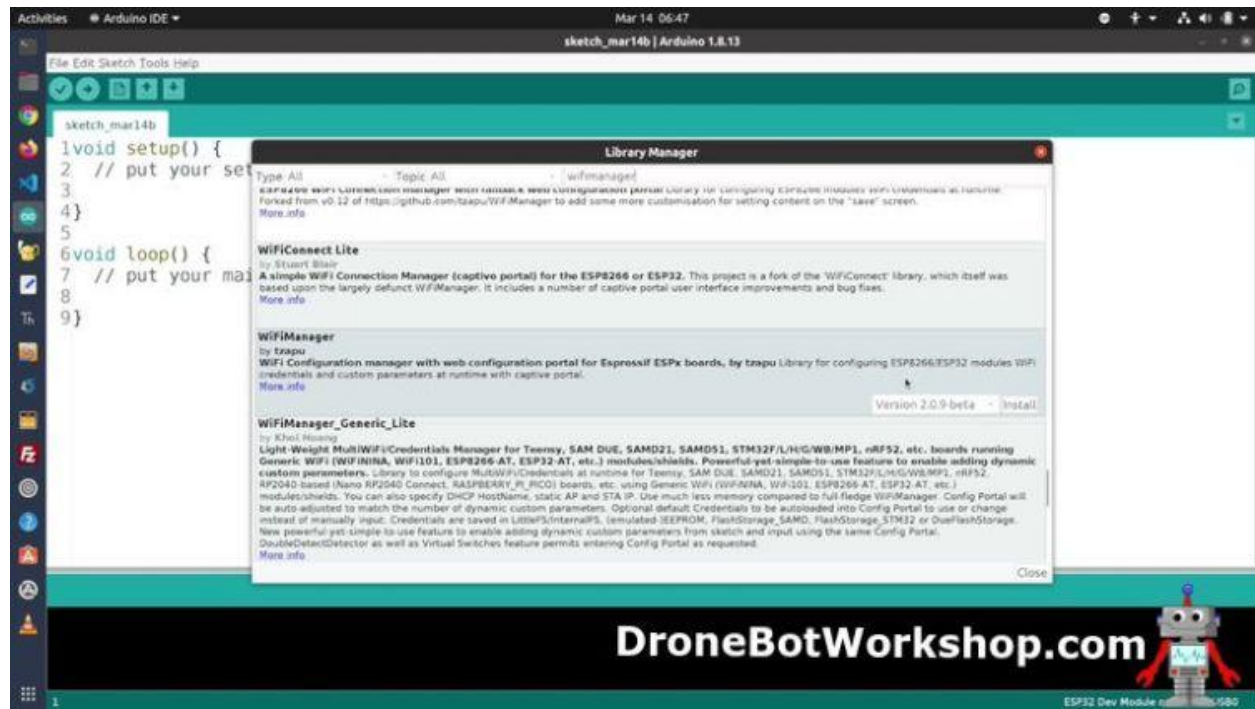
WiFiManager Installation

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

If you want to install the [WiFiManager from the GitHub repository](#), you can do that, but the easiest way to get it installed is to just use the Arduino IDE Library Manager.

Open your Library Manager and search for “WiFiManager” (all one word). You will get a bunch of results, scroll down to the “W” section and look for “WiFiManager by tzapu”. Click the “Install” button to add the library to your Arduino IDE.



To confirm that the installation was successful, open your Examples menu and scroll down to the section for “Examples from Custom Libraries”. Down near the bottom, you should see a “WiFiManager” selection. Highlight it, and you should see several example folders and files.

WiFiManager Demo

We will be testing out the installation using one of the example files installed when you added the WiFiManager library to your Arduino IDE.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Highlight the WiFiMaager item in the “Examples from Custom Libraries” section and look for the *Basic* example. Load that into your IDE.

```
1  #include <WiFiManager.h> // https://github.com/tzapu/WiFiManager
2
3
4  void setup() {
5      WiFi.mode(WIFI_STA); // explicitly set mode, esp defaults to STA+AP
6      // it is a good practice to make sure your code sets wifi mode how you want
7      it.
8
9      // put your setup code here, to run once:
10     Serial.begin(115200);
11
12     //WiFiManager, Local intialization. Once its business is done, there is no
13     need to keep it around
14     WiFiManager wm;
15
16     // reset settings - wipe stored credentials for testing
17     // these are stored by the esp library
18     //wm.resetSettings();
19
20     // Automatically connect using saved credentials,
21     // if connection fails, it starts an access point with the specified name (
22     "AutoConnectAP"),
23     // if empty will auto generate SSID, if password is blank it will be
24     anonymous AP (wm.autoConnect())
25     // then goes into a blocking loop awaiting configuration and will return
26     success result
27
28     bool res;
29
30     // res = wm.autoConnect(); // auto generated AP name from chipid
```

```
28     // res = wm.autoConnect("AutoConnectAP"); // anonymous ap
29     res = wm.autoConnect("AutoConnectAP","password"); // password protected ap
30
31     if(!res) {
32         Serial.println("Failed to connect");
33         // ESP.restart();
34     }
35     else {
36         //if you get here you have connected to the WiFi
37         Serial.println("connected...yeey :)");
38     }
39
40 }
41
void loop() {
    // put your main code here, to run repeatedly:
}
```

Let's take a look at the example, as it does a good job of illustrating how the WiFiManager library is used.

We begin by including the *WiFiManager* library.

The bulk of the sketch is inside the Setup function. We start setup by including the *WiFi* library, which is required to make WiFiManager work.

Next, we set up our serial monitor, which will be used to monitor the connection progress.

We build an object to represent our WiFiManager, in this sketch it is called “*wm*”, but you can use anything that makes sense to you.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

There is a command a bit further down, on line 15, that calls a *resetSettings()* method. By default, it is remarked out. If you run it, *resetSettings()* will wipe the current network configuration every time the ESP32 is booted up. You don't want to do this in production mode, but for testing, this is often useful.

We then define a boolean, called "res", to report the result of our WiFiManager connection attempt.

Below that, you can see the syntax for launching WiFiManager:

```
1 Autoconnect.(SP Name, Password)
```

The only required parameter is Autoconnect. The SP name is the name you want to give the local WiFi Access Point that is launched on the ESP32 when WiFiManager is started. If you leave out the password, then the local Access Point network won't require one.

After that, we look at the results and print them to the serial monitor.

As everything is run in Setup there is no code in the Loop.

Load the demo onto your ESP32 board and open up a WiFi-enabled device such as a phone, tablet, or computer. Look for the "AutoConnectAP" network, and connect to it using a password of "password".

Make sure you are running your serial monitor, and observe the debug information presented there.

After connecting, open a web browser and go to 192.168.4.1. On many phones and tablets, you will be automatically directed there.



You will see the WiFiManager opening screen. There are four sections on this screen:

- **Configure WiFi** – This is the button that you will want to select to connect your ESP32 to a WiFi network.
- **Info** – Some information about the ESP32 board.
- **Exit** – Exit the page.
- **Update** – OTA update of the ESP32 code.

Select *Configure WiFi*. There will be a short delay while the available WiFi networks are scanned. You can watch the progress on the serial monitor.

You will then be presented with a list of WiFi networks. It's a nice display, with signal strengths and protection status illustrated beside the network name.



Select the network you want to connect to, and its name will automatically populate the SSID text box. Use the other box to enter the network password, then press the Save button.

Watch the serial monitor. Assuming you entered the correct password, you should see the connection results.



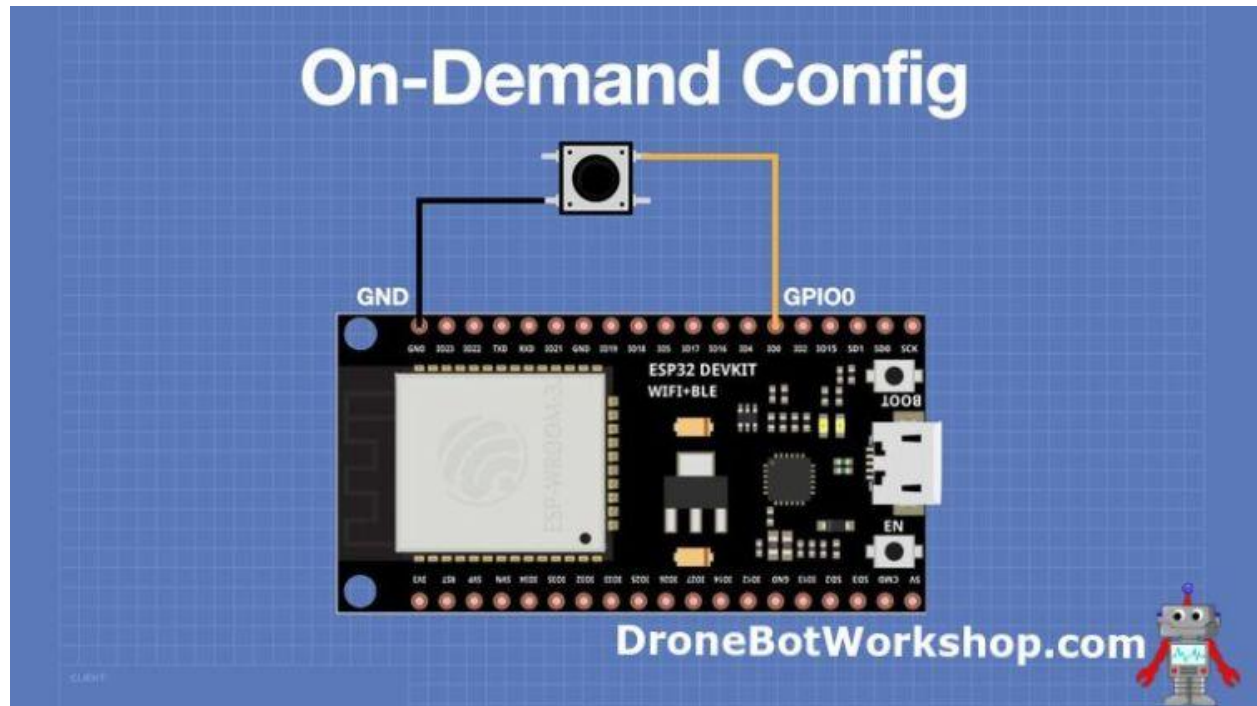
The ESP32 will now drop its local AP network and connect to the WiFi network. All without hard-coding any network credentials!

WiFiManager On Demand

By default, WiFiManager will only run when the ESP32 has no valid network configuration information. But there may be times when you want to run it on demand, such as when you move your ESP32 to a new location with a new WiFi network, or if you have multiple networks and wish to join another one.

You can use WiFiManager in an “on-demand” configuration. You’ll need to add a pushbutton to your ESP32, and you can then push it to invoke WiFiManager.

The following illustration shows a pushbutton wired between GPIO pin 0 and ground. You can use a different GPIO pin if pin 0 is not available.



The code for running WiFiManager on-demand is in another example included with the library. Go into the example code and look for the *OnDemand* entry. Open the only sketch in this entry, *OnDemandConfigPortal*.

The sketch is very similar to the first one, except for the fact that it runs in the Loop. We constantly check to see if the button has been pressed, and if it has, we run the WiFiManager as we did before.

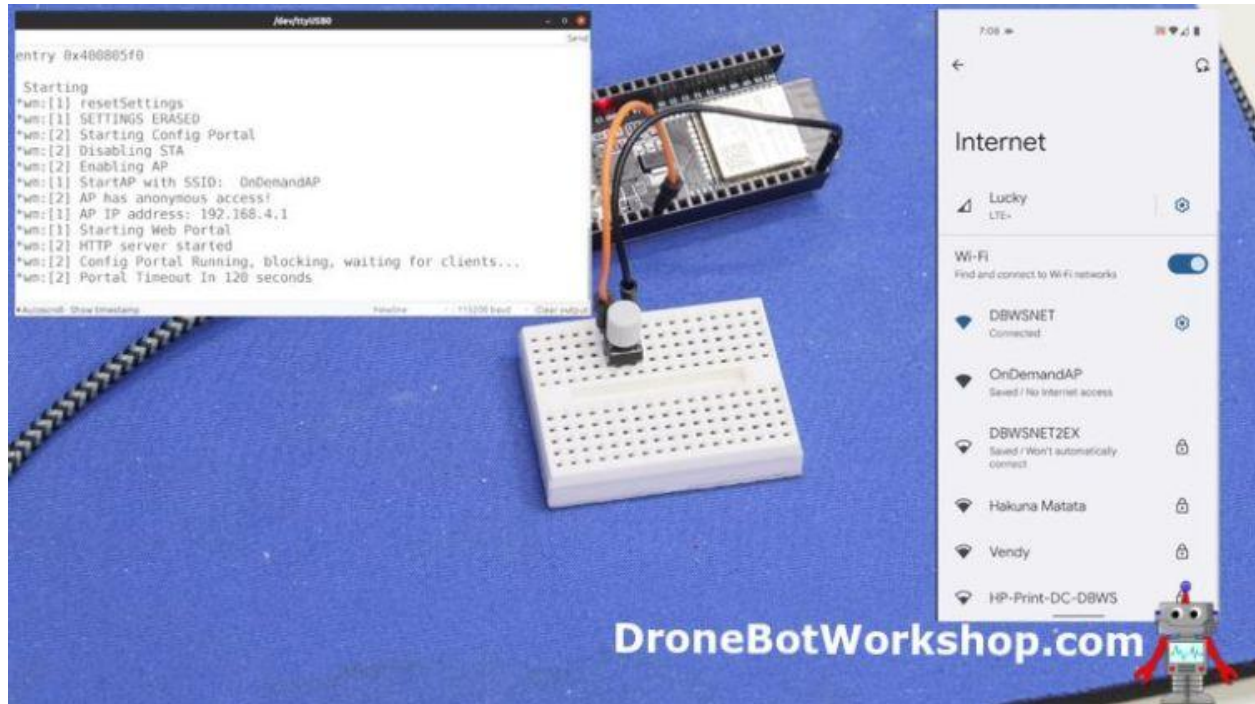

```
1  /**
2  * OnDemandConfigPortal.ino
3  * example of running the configPortal AP manually, independantly from the
4  * captiveportal
5  * trigger pin will start a configPortal AP for 120 seconds then turn it off.
6  *
7  */
8  #include <WiFiManager.h> // https://github.com/tzapu/WiFiManager
9
10 // select which pin will trigger the configuration portal when set to LOW
11 #define TRIGGER_PIN 0
12
13 int timeout = 120; // seconds to run for
14
15 void setup() {
16     WiFi.mode(WIFI_STA); // explicitly set mode, esp defaults to STA+AP
17     // put your setup code here, to run once:
18     Serial.begin(115200);
19     Serial.println("\n Starting");
20     pinMode(TRIGGER_PIN, INPUT_PULLUP);
21 }
22
23 void loop() {
24     // is configuration portal requested?
25     if ( digitalRead(TRIGGER_PIN) == LOW) {
26         WiFiManager wm;
27
28         //reset settings - for testing
29         //wm.resetSettings();
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
30 // set configportal timeout
31 wm.setConfigPortalTimeout(timeout);
32
33 if (!wm.startConfigPortal("OnDemandAP")) {
34     Serial.println("failed to connect and hit timeout");
35     delay(3000);
36     //reset and try again, or maybe put it to deep sleep
37     ESP.restart();
38     delay(5000);
39 }
40
41 //if you get here you have connected to the WiFi
42 Serial.println("connected...yeey :)");
43
44 }
45
46 // put your main code here, to run repeatedly:
47 }
```

If you use a different GPIO pin, just change the value of TRIGGER_PIN in the declarations section.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



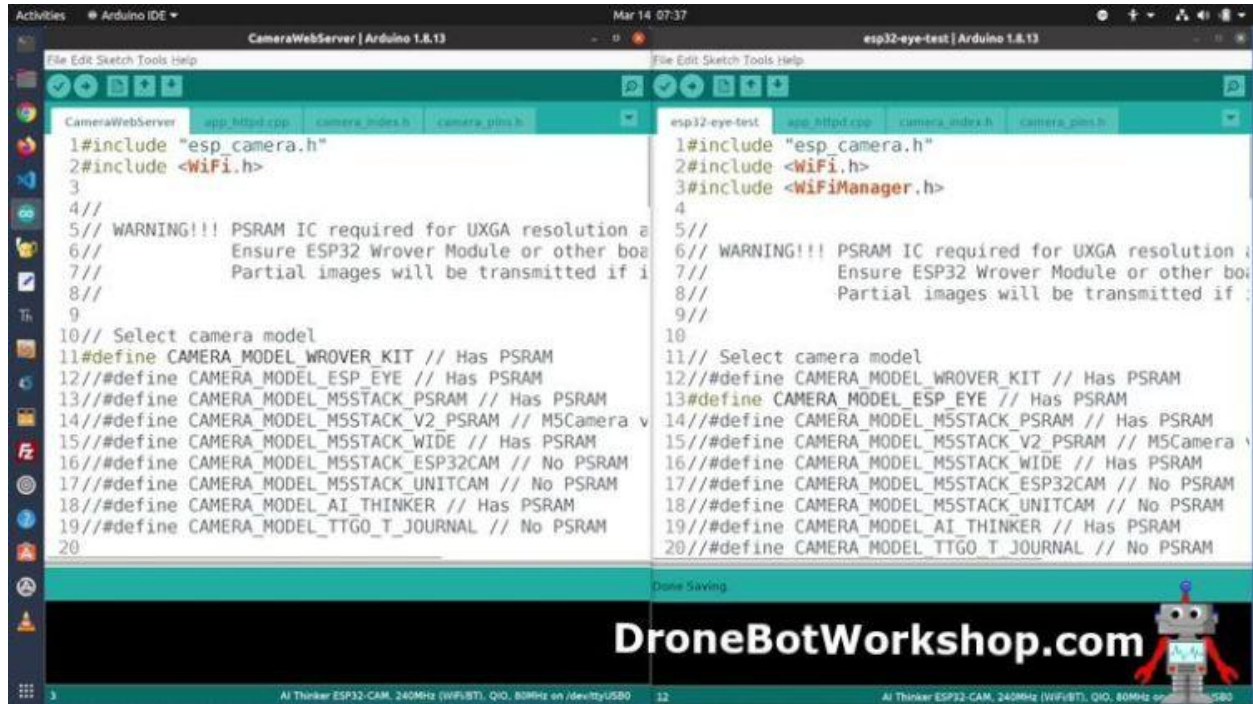
Load and run the sketch, with the serial monitor open. You should now be able to start the local access point, *OnDemandAP*, anytime by pressing the button.

Modifying Existing Sketches to use WiFiManager

We have now seen a couple of ways to run the WiFiManager utility. But how do we modify our own sketches to make use of it?

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



I'll show you exactly that, using a popular example sketch for the ESP32-CAM. In my case, I'll be using the ESP EYE, another ESP32 camera-equipped board. You can use the same principles to modify any ESP32 sketch.

Modifying the Camera Web Server Sketch

The *CameraWebServer* sketch is buried within the examples for the ESP32, in the *Camera* menu item. We have used this sketch before when we worked with the ESP32-CAM board. It's a complete camera control center and would be perfect for a remote camera setup.

All it needs is WiFiManager to make it complete!

Modifying the sketch is very easy:

1 – Include the WiFiManager library in the sketch.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  " >#include <WiFiManager.h>
```

2 – Replace the WiFi connection code as follows:

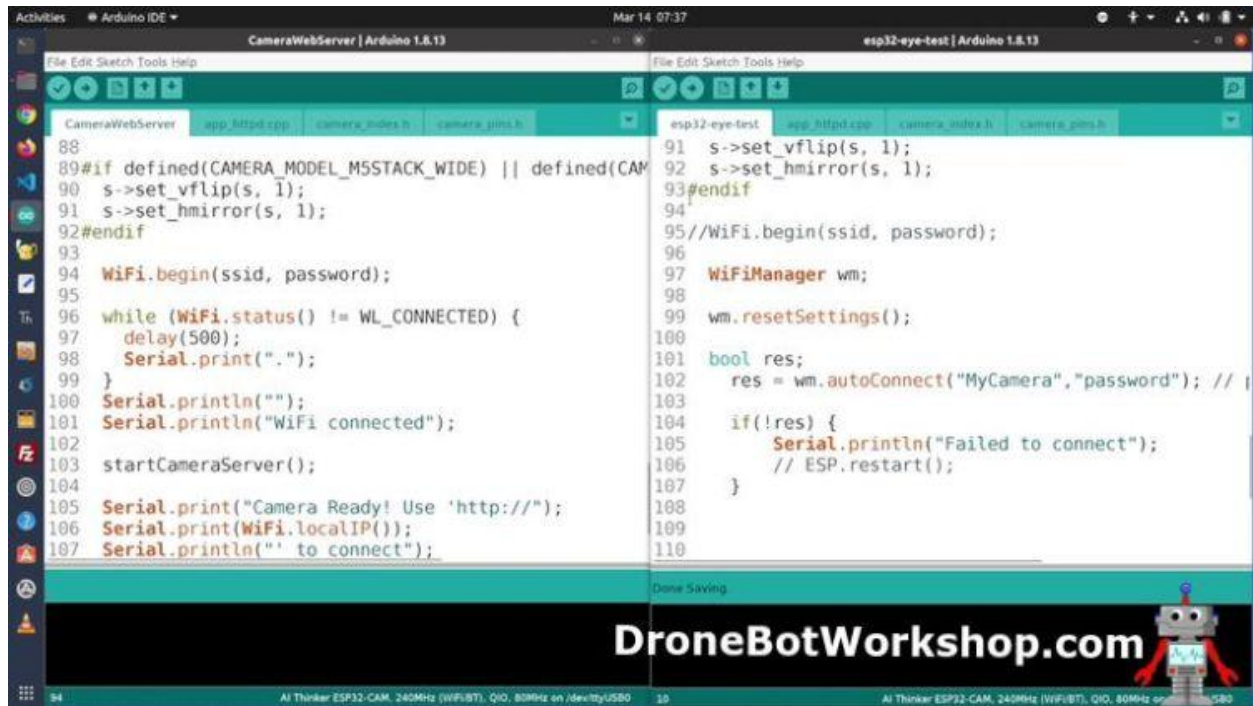
ORIGINAL:

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

REPLACEMENT:

```
1  WiFiManager wm;
2
3  bool res;
4
5  res = wm.autoConnect("AutoConnectAP","password"); // password protected ap
6
7
8
9
10  if(!res) {
11
12      Serial.println("Failed to connect");
13
14      // ESP.restart();
15
16  }
```

That's all you need to do! Your sketch will now work with WiFiManager, eliminating the need to hard-code the SSID and password.



Adding Custom Parameters

WiFiManager works great for obtaining your SSID and password information, but it can also be used to collect other configuration data as well.

This is perfect if you need to capture additional information, such as API keys, for your device to function.

Modifying the WiFiManager screen with additional text boxes is a 2-step process:

1 – You use the *WiFiManagerParameter* function to define your new text box.

2 – You add the parameter you've defined using the *addParameter* function.

The following code illustrates how this is accomplished:

```
1  /*
2   WiFiManager Extra Parameters Demo
3   wfm-parameter-demo.ino
4   Demonstrates adding extra parameters to WiFiManager menu
5
6   DroneBot Workshop 2022
7   https://dronebotworkshop.com
8  */
9
10 // Include WiFiManager Library
11 #include <WiFiManager.h>
12
13 void setup() {
14
15     // Setup Serial Monitor
16     Serial.begin(115200);
17
18     // Create WiFiManager object
19     WiFiManager wfm;
20
21     // Supress Debug information
22     wfm.setDebugOutput(false);
23
24     // Remove any previous network settings
25     wfm.resetSettings();
26
27     // Define a text box, 50 characters maximum
28     WiFiManagerParameter custom_text_box("my_text", "Enter your string here",
29     "default string", 50);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
30 // Add custom parameter
31 wfm.addParameter(&custom_text_box);
32
33 if (!wfm.autoConnect("ESP32TEST_AP", "password")) {
34     // Did not connect, print error message
35     Serial.println("failed to connect and hit timeout");
36
37     // Reset and try again
38     ESP.restart();
39     delay(1000);
40 }
41
42 // Connected!
43 Serial.println("WiFi connected");
44 Serial.print("IP address: ");
45 Serial.println(WiFi.localIP());
46
47
48 // Print custom text box value to serial monitor
49 Serial.print("Custom text box entry: ");
50 Serial.println(custom_text_box.getValue());
51 }
52
53 void loop() {
54
55     // Loop code
56
57 }
```

The line in which you add your parameter is this one:

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1 WiFiManagerParameter custom_text_box("my_text", "Enter your string here",  
    "default string", 50);
```

This breaks down as follows:

- **Parameter name** – *custom_text_box*
- **HTML element name** – *my_text*
- **Label for text box** – *Enter your string here*
- **Default text** – *default string*

We then add the parameter using this line:

```
1 wfm.AddParameter(&custom_text_box);
```

This sketch also illustrates how you can suppress debug information, using the following line:

```
1 wfm.setDebugOutput(false);
```

So you won't see as much information as you did in the previous tests.

Load the sketch onto an ESP32 and use your WiFi-enabled device to join the network and open WiFiManager. You should see the extra text field. Make sure that you also have your serial monitor open as well.

Replace the text with your own text string and add your network information. Save it, and observe the serial monitor.

You'll see your new IP address, plus the text that you entered into the new custom text box.

Saving WiFiManager Data

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Adding a text box to capture information is only one of the steps required to add extra configuration information. You'll also want to save that information on the ESP32 so that you don't need to enter it every time you power up or reset the device.

The ESP32 has some non-volatile flash memory that is ideal for saving configuration information. In order to use it, you will need to format it and store it using a file system.

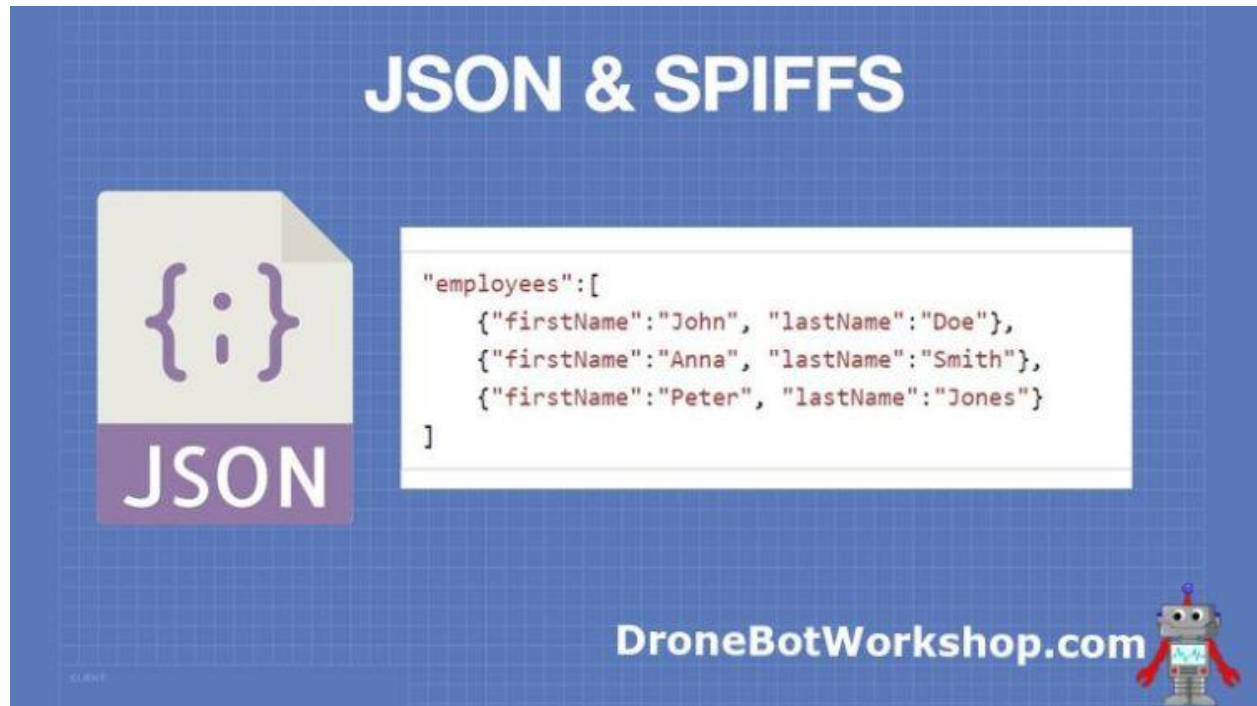
Let's see how to do both of those things.

JSON & SPIFFS

We will arrange our data in JSON format.

JSON, or JavaScript Object Notation, is a lightweight data-interchange format that is used in a multitude of applications. It is easily readable by both machines and humans and is used to interchange data between otherwise incompatible systems.

The format of a JSON document is illustrated here:



Now that we have our data arranged, we will need to have a method of storing it in the flash memory. This is where SPIFFS comes in.

SPIFFS is an abbreviation for SPI Flash File System. It is a file system used with NOR-gate embedded flash memory, which is the type of memory included on the ESP32.

SPIFFS allows you to treat the memory almost as you would treat SD or MicroSD card memory. I said “almost”, as there are a few differences. One difference is that SPIFFS does not support directories, although you can mimic them to some degree.

Flash memory has a limited number of write operations, so SPIFFS employs a number of techniques to maximize the lifespan of the memory. When the memory is formatted or erased, all the bits are set to one. When data is written to memory, only the bits that need to be pulled to zero are affected. SPIFFS also uses “wear leveling” to spread the data equally across the memory.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

By using JSON and SPIFFS we can store a good amount of configuration information in our ESP32, most ESP32 devices allocate about 1.5Mb for SPIFFS.

Saving Parameter Data

Let's take a look at a sketch that we can use to store our WiFiManager data into the ESP32's flash memory using JSON and SPIFFS:

```
1  /*
2  WiFiManager with saved textboxes Demo
3  wfm-text-save-demo.ino
4  Saves data in JSON file on ESP32
5  Uses SPIFFS
6
7  DroneBot Workshop 2022
8  https://dronebotworkshop.com
9
10 Functions based upon sketch by Brian Lough
11 https://github.com/witnessmenow/ESP32-WiFi-Manager-Examples
12 */
13
14 #define ESP_DRD_USE_SPIFFS true
15
16 // Include Libraries
17
18 // WiFi Library
19 #include <WiFi.h>
20
21 // File System Library
22 #include <FS.h>
23
24 // SPI Flash Syetem Library
25 #include <SPIFFS.h>
26
27 // WiFiManager Library
28 #include <WiFiManager.h>
29
30 // Arduino JSON library
31 #include <ArduinoJson.h>
32
33 // JSON configuration file
```

```
30 #define JSON_CONFIG_FILE "/test_config.json"
31
32 // Flag for saving data
33 bool shouldSaveConfig = false;
34
35 // Variables to hold data from custom textboxes
36 char testString[50] = "test value";
37 int testNumber = 1234;
38
39 // Define WiFiManager Object
40 WiFiManager wm;
41
42 void saveConfigFile()
43 // Save Config in JSON format
44 {
45     Serial.println(F("Saving configuration..."));
46
47     // Create a JSON document
48     StaticJsonDocument<512> json;
49     json["testString"] = testString;
50     json["testNumber"] = testNumber;
51
52     // Open config file
53     File configFile = SPIFFS.open(JSON_CONFIG_FILE, "w");
54     if (!configFile)
55     {
56         // Error, file did not open
57         Serial.println("failed to open config file for writing");
58     }
```

```
59
60 // Serialize JSON data to write to file
61 serializeJsonPretty(json, Serial);
62 if (serializeJson(json, configFile) == 0)
63 {
64     // Error writing file
65     Serial.println(F("Failed to write to file"));
66 }
67 // Close file
68 configFile.close();
69 }
70
71 bool loadConfigFile()
72 // Load existing configuration file
73 {
74     // Uncomment if we need to format filesystem
75     // SPIFFS.format();
76
77     // Read configuration from FS json
78     Serial.println("Mounting File System...");
79
80     // May need to make it begin(true) first time you are using SPIFFS
81     if (SPIFFS.begin(false) || SPIFFS.begin(true))
82     {
83         Serial.println("mounted file system");
84         if (SPIFFS.exists(JSON_CONFIG_FILE))
85         {
86             // The file exists, reading and loading
87             Serial.println("reading config file");
```

```
88     File configFile = SPIFFS.open(JSON_CONFIG_FILE, "r");
89     if (configFile)
90     {
91         Serial.println("Opened configuration file");
92         StaticJsonDocument<512> json;
93         DeserializationError error = deserializeJson(json, configFile);
94         serializeJsonPretty(json, Serial);
95         if (!error)
96         {
97             Serial.println("Parsing JSON");
98
99             strcpy(testString, json["testString"]);
100             testNumber = json["testNumber"].as<int>();
101
102             return true;
103         }
104         else
105         {
106             // Error loading JSON data
107             Serial.println("Failed to load json config");
108         }
109     }
110 }
111 else
112 {
113     // Error mounting file system
114     Serial.println("Failed to mount FS");
115 }
116 }
```



```
11
1
    return false;
11
2
}
11
3
11
4
void saveConfigCallback()
// Callback notifying us of the need to save configuration
11
5
{
11
    Serial.println("Should save config");
6
    shouldSaveConfig = true;
11
7
}
11
8
void configModeCallback(WiFiManager *myWiFiManager)
// Called when config mode launched
11
9
{
12
0
    Serial.println("Entered Configuration Mode");
12
1
    Serial.print("Config SSID: ");
12
2
    Serial.println(myWiFiManager->getConfigPortalSSID());
12
3
    Serial.print("Config IP Address: ");
12
4
    Serial.println(WiFi.softAPIP());
12
5
}
12
6
void setup()
{
12
7
    // Change to true when testing to force configuration every time we run
12
8
    bool forceConfig = false;
```

```
12  bool spiffsSetup = loadConfigFile();
9
13  if (!spiffsSetup)
0  {
13      Serial.println(F("Forcing config mode as there is no saved config"));
1      forceConfig = true;
13  }
2
13
3  // Explicitly set WiFi mode
13  WiFi.mode(WIFI_STA);
4
13
5  // Setup Serial monitor
13  Serial.begin(115200);
6
13  delay(10);
7
13  // Reset settings (only for development)
8  wm.resetSettings();
13
9
14  // Set config save notify callback
0  wm.setSaveConfigCallback(saveConfigCallback);
14
1  // Set callback that gets called when connecting to previous WiFi fails, and
14  enters Access Point mode
2  wm.setAPCallback(configModeCallback);
14
3
14  // Custom elements
4
14
5  // Text box (String) - 50 characters maximum
14  WiFiManagerParameter custom_text_box("key_text", "Enter your string here",
6  testString, 50);
```

```
14 // Need to convert numerical input to string to display the default value.
7 char convertedValue[6];
14 sprintf(convertedValue, "%d", testNumber);
8
14
9 // Text box (Number) - 7 characters maximum
15 WiFiManagerParameter custom_text_box_num("key_num", "Enter your number here",
0 convertedValue, 7);
15
1 // Add all defined parameters
15 // Add all defined parameters
2 wm.addParameter(&custom_text_box);
15 wm.addParameter(&custom_text_box_num);
3
15
4 if (forceConfig)
15 // Run if we need a configuration
5 {
15 if (!wm.startConfigPortal("NEWTEST_AP", "password"))
6 {
15 Serial.println("failed to connect and hit timeout");
7
15 delay(3000);
8
15 //reset and try again, or maybe put it to deep sleep
15 ESP.restart();
9
16 delay(5000);
0
1 }
16 }
1
else
16 {
2
16 if (!wm.autoConnect("NEWTEST_AP", "password"))
3 {
16 Serial.println("failed to connect and hit timeout");
4
delay(3000);
```

```
16      // if we still have not connected restart and try all over again
5
    ESP.restart();
16
6      delay(5000);
16
    }
7
}

16
8
// If we get here, we are connected to the WiFi
16
9

17    Serial.println("");
0
    Serial.println("WiFi connected");
17
1    Serial.print("IP address: ");
17
    Serial.println(WiFi.localIP());
2

17
3    // Lets deal with the user config values
17
4
    // Copy the string value
17    strncpy(testString, custom_text_box.getValue(), sizeof(testString));
5
    Serial.print("testString: ");
17
6    Serial.println(testString);
17
7
    //Convert the number value
17
8    testNumber = atoi(custom_text_box_num.getValue());
17
    Serial.print("testNumber: ");
9
    Serial.println(testNumber);
18
0
18
1    // Save the custom parameters to FS
18
2    if (shouldSaveConfig)
    {
```

```
18     saveConfigFile();  
3
```

```
18  
4 }
```

```
18  
5
```

```
18  
6
```

```
18  
7 void loop() {
```

```
18     // put your main code here, to run repeatedly:  
8
```

```
18  
9 }
```

```
19  
0
```

```
19  
1
```

```
19  
2
```

```
19  
3
```

```
19  
4
```

```
19  
5
```

```
19  
6
```

```
19  
7
```

```
19  
8
```

```
19  
9
```

```
20  
0
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

20

1

20

2

20

3

20

4

20

5

20

6

20

7

20

8

20

9

21

0

21

1

21

2

21

3

21

4

21

5

21

6

21

7

21

8

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

21

9

22

0

22

1

22

2

22

3

22

4

22

5

22

6

22

7

22

8

22

9

23

0

23

1

23

2

23

3

23

4

23

5

23

6

<https://dronebotworkshop.com>

```
23  
7  
23  
8  
23  
9  
24  
0  
24  
1  
24  
2
```

The functions used in the sketch are not mine, they were adapted from some of the excellent work done by [Brian Lough on GitHub](#). You should visit his GitHub page for many more examples, including ones that use checkboxes and drop-down menus.

You'll need to install the Arduino JSON library, you can get this using the Arduino IDE Library Manager. Search for "Arduino JSON" and install the library from Benoit Blanchon.

The SPIFFS and FS (File System) libraries will already have been installed when you added the ESP32 boards manager to your Arduino IDE.

We will be setting up two custom text boxes, one that accepts text and one that only accepts numbers. Note that the data is stored as a character variable array, so you need to convert the numbers to characters (and convert them back) to use them.

When you run the sketch, be sure to have your serial monitor open. Observe the monitor after saving the data, you'll see it in its JSON format.

Now power off the ESP32 and then power it back up. Go back into the WiFiManager and look at the custom text boxes. They should still show the data you previously entered.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Using this technique, you can capture and store custom parameters, as well as WiFi information, for your ESP32.

And no hard-coding!

Conclusion

WiFiManager is like the “missing link” for the ESP32. By using it, we can create professional-level personal projects or even commercial devices.

This would be perfect for IoT projects, or for anything that needs to be able to connect to any WiFi network.

Give it a try. Once you do, I’m sure your days of hard-coding WiFi credentials will have come to an end!