TASK: ESP BLE Provisioning

#### **Description**

This project demonstrates how to provision an ESP32 to a Wi-Fi network using Bluetooth Low Energy (BLE). It uses the **ESP32’s BLE capability** to allow users to connect to the device and input Wi-Fi credentials via a mobile app, making the Wi-Fi setup seamless and easy. Once connected, the device stores the Wi-Fi credentials in non-volatile storage, allowing it to reconnect automatically even after a power cycle. This is particularly useful in environments where the device may be deployed without direct access to update the Wi-Fi credentials.

The provisioning process leverages the **Espressif Wi-Fi Provisioning Manager** library to handle BLE interactions securely and efficiently. Users can use the **Espressif Provisioning App** on their smartphone to enter Wi-Fi credentials, which are then sent to the ESP32 over BLE. This code is compatible with the **Arduino IDE** and requires the **WiFiProv** library, which can be installed through the Arduino Library Manager.

#### **Required Libraries**

* **WiFiProv**: Manages BLE provisioning
* **WiFi**: Handles Wi-Fi connections

### **Problem Solved by This Task**

Setting up Wi-Fi connectivity on IoT devices can be challenging, particularly if the device is installed in hard-to-reach places or lacks an accessible interface. BLE provisioning solves this problem by allowing the device to advertise itself over BLE so users can connect with a mobile device to input Wi-Fi credentials. This is:

* **Efficient**: Reduces the need for physical setup interfaces like screens or keyboards.
* **User-Friendly**: BLE is a common feature on smartphones, making the setup straightforward.
* **Secure**: Uses Proof of Possession (PoP) to prevent unauthorized access during provisioning.

#### **Code Overview**

#include "WiFiProv.h"

#include "WiFi.h"

const char \*pop = "abcd1234";

const char \*service\_name = "PROV\_123";

const char \*service\_key = NULL;

bool reset\_provisioned = true;

void SysProvEvent(arduino\_event\_t \*sys\_event) {

switch (sys\_event->event\_id) {

case ARDUINO\_EVENT\_WIFI\_STA\_GOT\_IP:

Serial.print("\nConnected! IP address: ");

Serial.println(IPAddress(sys\_event->event\_info.got\_ip.ip\_info.ip.addr));

break;

case ARDUINO\_EVENT\_WIFI\_STA\_DISCONNECTED:

Serial.println("\nDisconnected. Reconnecting...");

break;

case ARDUINO\_EVENT\_PROV\_START:

Serial.println("\nProvisioning started\nEnter Wi-Fi credentials using the smartphone app.");

break;

case ARDUINO\_EVENT\_PROV\_CRED\_RECV: {

Serial.println("\nReceived Wi-Fi credentials:");

Serial.print("\tSSID: ");

Serial.println((const char \*)sys\_event->event\_info.prov\_cred\_recv.ssid);

Serial.print("\tPassword: ");

Serial.println((const char \*)sys\_event->event\_info.prov\_cred\_recv.password);

break;

}

case ARDUINO\_EVENT\_PROV\_CRED\_FAIL: {

Serial.println("\nProvisioning failed. Try resetting to factory settings.");

if (sys\_event->event\_info.prov\_fail\_reason == WIFI\_PROV\_STA\_AUTH\_ERROR) {

Serial.println("Incorrect Wi-Fi password.");

} else {

Serial.println("Wi-Fi AP not found. Try erasing NVS with 'nvs\_flash\_erase()' before beginProvision().");

}

break;

}

case ARDUINO\_EVENT\_PROV\_CRED\_SUCCESS:

Serial.println("\nProvisioning Successful! Device is now connected.");

break;

case ARDUINO\_EVENT\_PROV\_END:

Serial.println("\nProvisioning ended.");

break;

default:

break;

}

}

void setup() {

Serial.begin(115200);

WiFi.onEvent(SysProvEvent);

Serial.println("Starting BLE Provisioning...");

uint8\_t uuid[16] = {

0xb4, 0xdf, 0x5a, 0x1c, 0x3f, 0x6b, 0xf4, 0xbf,

0xea, 0x4a, 0x82, 0x03, 0x04, 0x90, 0x1a, 0x02

};

WiFiProv.beginProvision(WIFI\_PROV\_SCHEME\_BLE,

WIFI\_PROV\_SCHEME\_HANDLER\_FREE\_BTDM,

WIFI\_PROV\_SECURITY\_1,

pop,

service\_name,

service\_key,

uuid,

reset\_provisioned);

Serial.println("BLE QR Code for provisioning:");

WiFiProv.printQR(service\_name, pop, "ble");

}

void loop() {

}

**Explanation of the working principle of code**

1. **Provisioning Parameters**:
   * pop: Proof of Possession (PoP) is a password required to authenticate during the provisioning process.
   * service\_name: Sets the Bluetooth device name, visible to the app.
   * service\_key: Password for SoftAP (not required for BLE provisioning).
   * reset\_provisioned: Erases previous provisioning data to ensure new credentials are used.
2. **Event Handler (SysProvEvent)**:
   * Handles various events during the provisioning process, such as Wi-Fi connection, credential receipt, provisioning failure, and successful connection.
   * Outputs relevant logs to the Serial Monitor, which can be helpful for debugging.
3. **BLE Provisioning Setup**:
   * The UUID used allows the app to recognize the device.
   * beginProvision() initializes provisioning with BLE, using security mode 1 and a predefined PoP.
4. **QR Code for Provisioning**:
   * A QR code with the device name and PoP is generated, allowing users to quickly connect to the device via the provisioning app by scanning the code.

### **Setup Instructions**

1. **Install Dependencies**:
   * Ensure the **WiFiProv** library is installed in the Arduino IDE.
   * Install the **Espressif Provisioning App** on your smartphone to complete the provisioning.
2. **Upload the Code**:
   * Select the appropriate partition scheme (e.g., “No OTA (2MB APP/2MB SPIFFS)”) in Arduino IDE under Tools > Partition Scheme.
3. **Provisioning**:
   * Open the Serial Monitor at 115200 baud rate to observe the logs.
   * Scan for your device in the app, enter the Wi-Fi credentials, and initiate provisioning.
4. **Monitor Output**:
   * Logs in the Serial Monitor will indicate the provisioning progress, any connection errors, or successful provisioning events.

### **Troubleshooting**

* **Provisioning Fails**:
  + If provisioning fails, check the logs for errors.
  + You may need to call nvs\_flash\_erase() to clear previously saved data if you notice repeated failures due to incorrect credentials or connection issues.
* **Device Not Visible in App**:
  + Verify the ESP32 has BLE enabled and ensure your smartphone's Bluetooth is active.
  + Ensure the BLE UUID matches the format expected by the provisioning app.

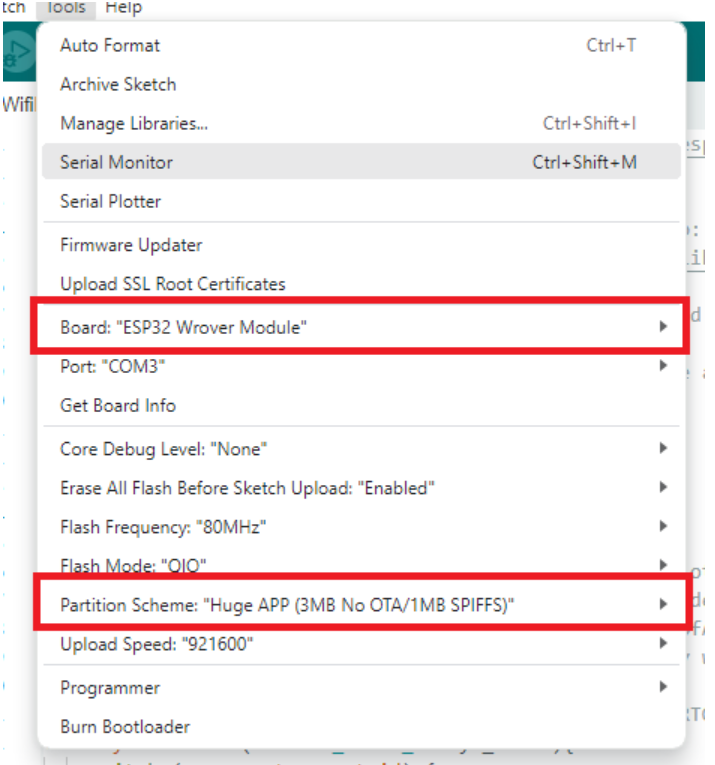
**Instructions for Uploading the Code**

This code is memory-intensive, which may prevent it from flashing with default settings on some ESP32 boards. If you encounter a “Sketch too big” error during the upload, you’ll need to modify the partition scheme in the Arduino IDE:

1. Go to \*\*Tools > Partition Scheme\*\*.

2. Select a scheme that has at least \*\*1.4MB of available space for the APP\*\*. For example, try “\*\*Huge APP (3MB No OTA/1MB SPIFFS)\*\*.”

If the \*\*Partition Scheme\*\* option is not visible under the \*\*Tools\*\* menu, switch to a different board model such as the \*\*ESP32 Wrover Module\*\*, which supports larger partition sizes.



**Testing the Code**

To ensure the ESP32 Wi-Fi provisioning works correctly, follow these steps:

1. \*\*Upload the Code\*\*: Once you have uploaded the code to the ESP32, open the \*\*Serial Monitor\*\* in the Arduino IDE, setting the baud rate to \*\*115200\*\*.

2. \*\*Start the Program\*\*: Press the \*\*RST (Reset) button\*\* on the ESP32 to initiate the code execution.

**\*\*Note\*\***: You may encounter a PSRAM error on the Serial Monitor. This error is safe to ignore, as it does not impact the Wi-Fi provisioning process.

3. \*\*Observe Serial Monitor Logs\*\*: In the Serial Monitor, you should see output messages confirming that the BLE provisioning process has started. The logs will resemble:

```

Begin Provisioning using BLE

Provisioning started

Provide Wi-Fi credentials using the smartphone app

```



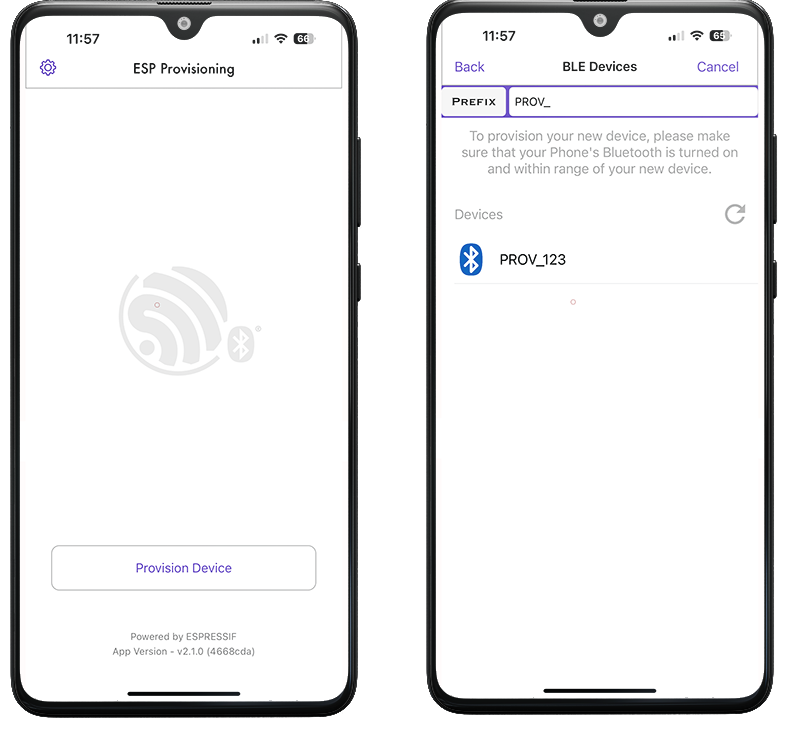
**Connecting with the Espressif Wi-Fi Provisioning App**

4. \*\*Launch the Espressif Wi-Fi Provisioning App\*\*: Open the Espressif Wi-Fi Provisioning App on your smartphone.

5. \*\*Provision Device\*\*:

- In the app, select \*\*Provision Device\*\*.

- The app will display an option to \*\*Scan QR Code\*\*. If this option does not work, select \*\*I don’t have a QR code\*\*.



6. \*\*Select ESP32 Device\*\*:

- A list of available Bluetooth devices with the prefix “\*\*PROV\_\*\*” will appear. In this case, the ESP32 device is configured to appear as \*\*“PROV\_123”\*\*. Select this device to proceed.

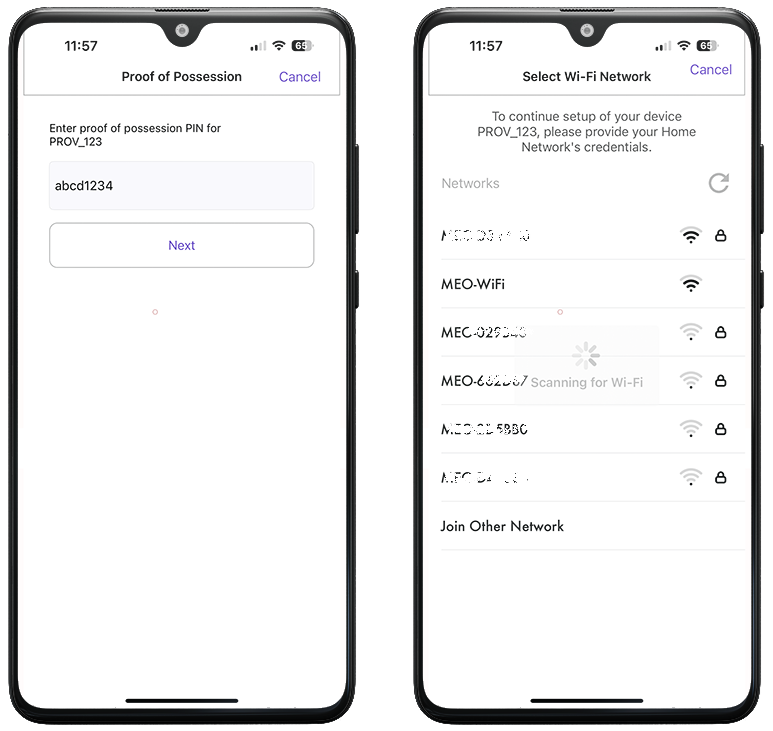
7. \*\*Enter Proof of Possession (PoP)\*\*:

- When prompted, enter the Proof of Possession (PoP) value defined in the code. For this setup, the default PoP is \*\*abcd1234\*\*. Make sure this value matches the one in the code exactly.

8. \*\*Select Wi-Fi Network\*\*:

- After successful authentication, a list of nearby Wi-Fi networks will appear.

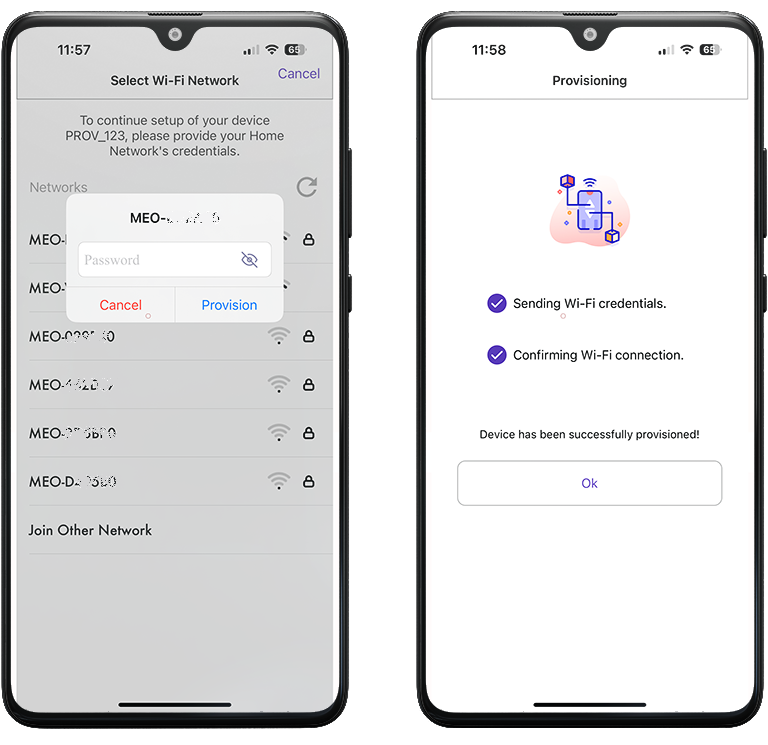
- If your network does not appear, you can manually add it by selecting \*\*Join Other Network\*\*.



9. \*\*Enter Wi-Fi Credentials\*\*:

- Enter the password for your Wi-Fi network and tap \*\*Connect\*\*.

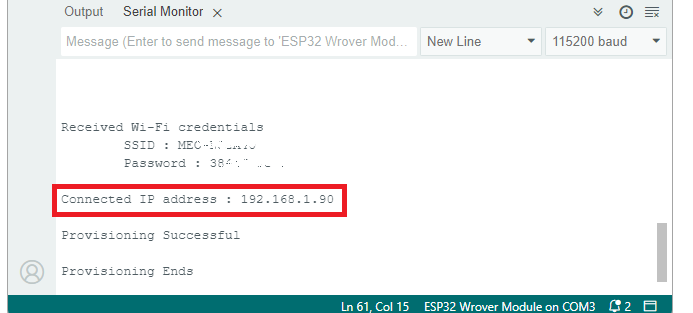
- After a few seconds, the Wi-Fi credentials will be sent to the ESP32, allowing it to connect to the network.



10. \*\*Confirm Connection\*\*:

- In the \*\*Serial Monitor\*\*, you should see a confirmation message that the ESP32 has received the Wi-Fi credentials and has connected to the network successfully.

- The ESP32 will display its assigned \*\*IP address\*\* on the network.



Wi-Fi provisioning is now successfully implemented on the ESP32, allowing it to connect seamlessly to a network using the Espressif provisioning app.

### **Conclusion**

This project efficiently provisions an ESP32 to a Wi-Fi network using BLE, providing a simple, user-friendly solution for Wi-Fi configuration in IoT applications. The code is designed to be flexible and can be expanded for use in larger systems requiring Wi-Fi-based device control and monitoring.