Part 24
-
Saving Data in Flash Memory

This guide shows how to save data permanently on the ESP32 flash memory using the Preferences.h library. The data held in the flash memory persists across resets or power failures. Using thePreferences.h library is useful to save data like network credentials, API keys, threshold values, or even the last state of a GPIO. You'll learn how to save and read data from flash memory.

Preferences.h Library
This library is "installed" automatically when you install the ESP32 boards in your Arduino IDE. The Preferences.h library is preferably used to store variable values through key:value pairs. Saving data permanently can be important to:

- remember the last state of a variable;
- save settings;
- save how many times an appliance was activated;
- or any other data type you need to save permanently.

If, instead of variables, you need to save files on the ESP32, we recommend using the filesystem (SPIFFS) instead.

Save Data Using Preferences.h Library
The data saved using Preferences is structured like this:

```
namespace {
  key:value
}
```

You can save different keys on the same namespace, for example:

```
namespace {
  key1: value1
  key2: value2
}
```

In a practical example, this configuration could be used to save your network credentials:

```
credentials {
  ssid: "your_ssid"
  pass: "your_pass"
}
```

In the preceding example, credentials is the namespace, and ssid and pass are the keys.
You can also have multiple namespaces with the same key; each key with its value:

```
namespace1{
  key:value1
}
namespace2{
  key:value2
}
```

When using the Preferences.h library, you should define the data type you want to save. Later, if you want to read that data, you must know the saved data type. In other words, the data type of writing and reading should be the same.
You can save the following data types using Preferences.h: char, Uchar, short, Ushort, int, Uint, long, Ulong, long64, Ulong64, float, double, bool, string and bytes.
For more information, you can access the Preferences.cpp file.

Preferences.h Library Useful Functions
To use the Preferences.h library to store data, first you need to include it in your sketch:

```
#include <Preferences.h>
```

Then, you must initiate an instance of the Preferences library. You can call it `preferences`, for example:

```
Preferences preferences;
```

After this, you can use the following methods to handle data using the Preferences.h library.

Start Preferences
The `begin()` method opens a "storage space" with a defined namespace. The `false` argument means that we'll use it in read/write mode. Use `true` to open or create the namespace in read-only mode.

```
preferences.begin("my-app", false);
```

In this case, the namespace name is `my-app`. Namespace name is limited to 15 characters.

Clear Preferences
Use `clear()` to clear all preferences under the opened namespace (it doesn't delete the namespace):

```
preferences.clear();
```

Remove Key
Remove a key from the opened namespace:

```
preferences.remove(key);
```

Close Preferences
Use the `end()` method to close the preferences under the opened namespace:

```
preferences.end();
```

Put a Key Value (Save a value)
You should use different methods depending on the variable type you want to save.

Char	putChar(const char* key, int8_t value)
Unsigned Char	putUChar(const char* key, int8_t value)
Short	putShort(const char* key, int16_t value)
Unsigned Short	putUShort(const char* key, uint16_t value)
Int	putInt(const char* key, int32_t value)
Unsigned Int	putUInt(const char* key, uint32_t value)
Long	putLong(const char* key, int32_t value)
Unsigned Long	putULong(const char* key, uint32_t value)
Long64	putLong64(const char* key, int64_t value)
Unsigned Long64	putULong64(const char* key, uint64_t value)
Float	putFloat(const char* key, const float_t value)
Double	putDouble(const char* key, const double_t value)
Bool	putBool(const char* key, const bool value)
String	putString(const char* key, const String value)
Bytes	putBytes(const char* key, const void* value, size_t len)

Get a Key Value (Read Value)
Similarly, you should use different methods depending on the variable type you want to get.

Char	getChar(const char* key, const int8_t defaultValue)
Unsigned Char	getUChar(const char* key, const uint8_t defaultValue)
Short	getShort(const char* key, const int16_t defaultValue
Unsigned Short	getUShort(const char* key, const uint16_t defaultValue)
Int	getInt(const char* key, const int32_t defaultValue)
Unsigned Int	getUInt(const char* key, const uint32_t defaultValue)
Long	getLong(const char* key, const int32_t defaultValue)
Unsigned Long	getULong(const char* key, const uint32_t defaultValue)
Long64	getLong64(const char* key, const int64_t defaultValue)
Unsigned Long64	gettULong64(const char* key, const uint64_t defaultValue)
Float	getFloat(const char* key, const float_t defaultValue)
Double	getDouble(const char* key, const double_t defaultValue)
Bool	getBool(const char* key, const bool defaultValue)
String	getString(const char* key, const String defaultValue)
String	getString(const char* key, char* value, const size_t maxLen)
Bytes	getBytes(const char* key, void * buf, size_t maxLen)

Remove a Namespace
In the Arduino implementation of `Preferences`, there is no method of completely removing a namespace. As a result, over the course of several projects, the ESP32 non-volatile storage (nvs) `Preferences` partition may become full. To completely erase and reformat the NVS memory used by Preferences, create a sketch that contains:

```
#include <nvs_flash.h>

void setup() {
  nvs_flash_erase(); // erase the NVS partition and...
  nvs_flash_init(); // initialize the NVS partition.
  while(true);
}

void loop() {

}
```

You should download a new sketch to your board immediately after running the above, or it will reformat the NVS partition every time it is powered up.

Example: Save key:value Pairs

For a simple example on how to save and get data using `Preferences.h`, in your Arduino IDE, go to `File > Examples > Preferences > StartCounter`.

```
#include <Preferences.h>

Preferences preferences;

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

  // Open Preferences with my-app namespace. Each application module, library, etc
  // has to use a namespace name to prevent key name collisions. We will open storage in
  // RW-mode (second parameter has to be false).
  // Note: Namespace name is limited to 15 chars.
  preferences.begin("my-app", false);

  // Remove all preferences under the opened namespace
  //preferences.clear();

  // Or remove the counter key only
  //preferences.remove("counter");

  // Get the counter value, if the key does not exist, return a default value of 0
  // Note: Key name is limited to 15 chars.
  unsigned int counter = preferences.getUInt("counter", 0);

  // Increase counter by 1
  counter++;

  // Print the counter to Serial Monitor
  Serial.printf("Current counter value: %u\n", counter);

  // Store the counter to the Preferences
  preferences.putUInt("counter", counter);

  // Close the Preferences
  preferences.end();

  // Wait 10 seconds
  Serial.println("Restarting in 10 seconds...");
  delay(10000);

  // Restart ESP
  ESP.restart();
}

void loop() {

}
```

This example increases a variable called `counter` between resets. This illustrates that the ESP32 “remembers” the value even after a reset.

Upload the previous sketch to your ESP32 board. Open the Serial Monitor at a baud rate of 115200 and press the on-board RST button. You should see the `counter` variable increasing between resets.

How the Code Works

This example uses the functions we've seen in the previous sections.
First, include the Preferences.h library.

```
#include <Preferences.h>
```

Then, create an instance of the library called preferences.
Preferences preferences;
In the `setup()`, initialize the Serial Monitor at a baud rate of 115200.

```
Serial.begin(115200);
```

Create a "storage space" in the flash memory called my-app in read/write mode. You can give it any other name.

```
preferences.begin("my-app", false);
```

Get the value of the counter key saved on preferences. If it doesn't find any value, it returns 0 by default (which happens when this code runs for the first time).

```
unsigned int counter = preferences.getUInt("counter", 0);
```

The counter variable is increased one unit every time the ESP runs:

```
counter++;
```

Print the value of the counter variable:

```
Serial.printf("Current counter value: %u\n", counter);
```

Store the new value on the “counter” key:

```
preferences.putUInt("counter", counter);
```

Close the Preferences.

```
preferences.end();
```

Finally, restart the ESP32 board:

```
ESP.restart();
```

Example - Save/Read Network Credentials using the Preferences.h Library
The Preferences.h library is many times used to save your network credentials permanently on the flash memory. This way, you don't have to hard code the credentials in every sketch that involves connecting the ESP32 to the internet.
The following sketch saves your network credentials permanently on the ESP32 flash memory using Preferences.h.

```
#include <Preferences.h>

Preferences preferences;

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

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

  preferences.begin("credentials", false);
  preferences.putString("ssid", ssid);
  preferences.putString("password", password);

  Serial.println("Network Credentials Saved using Preferences");

  preferences.end();
}

void loop() {

}
```

Don't forget to insert your network credentials in the following variables:

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

How the Code Works
Let's take a quick look at the relevant parts of the code for this example.
In the `setup()`, create a new storage space on the flash memory with the credentials namespace.

```
preferences.begin("credentials", false);
```

Then, create a key called ssid that saves your SSID value (ssid variable) – use the putString() method.

```
preferences.putString("ssid", ssid);
```

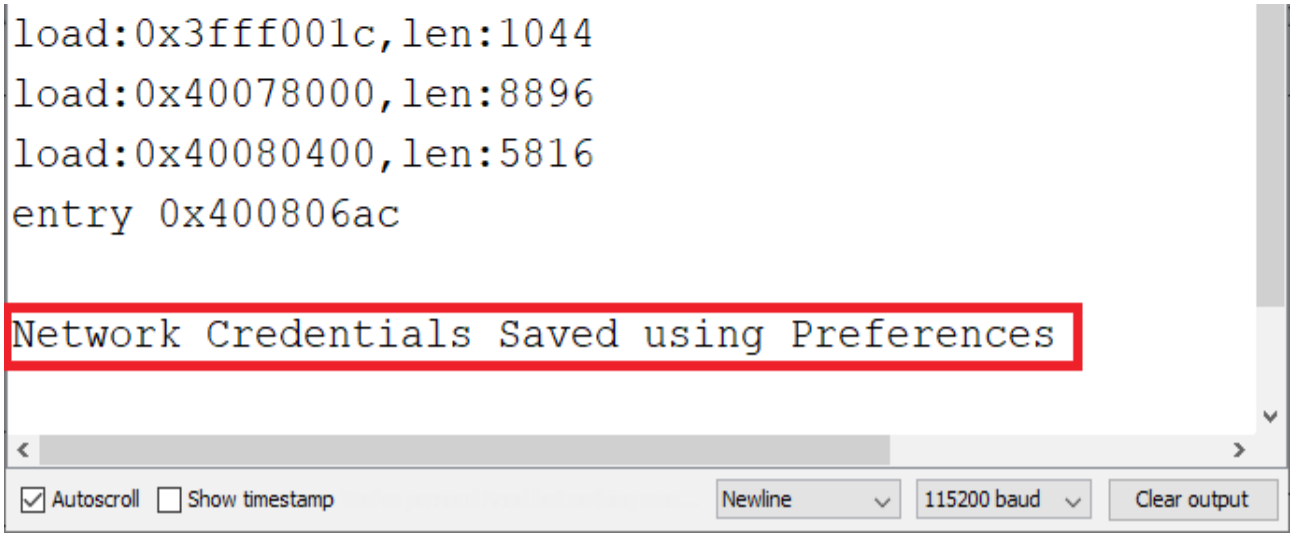
Add another key called password to save the password value (password variable):

```
preferences.putString("password", password);
```

So, your data is structured in this way:

```
credentials{
  ssid: your_ssid
  password: your_password
}
```

Upload the code to your board and this is what you should get on the Serial Monitor:



Connect to Wi-Fi with Network Credentials Saved on Preferences
The following sketch gets the network credentials' values and connects to your network using those credentials.

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

Preferences preferences;

String ssid;
String password;

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

  preferences.begin("credentials", false);

  ssid = preferences.getString("ssid", "");
  password = preferences.getString("password", "");

  if (ssid == "" || password == ""){
    Serial.println("No values saved for ssid or password");
  }
  else {
    // Connect to Wi-Fi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid.c_str(), password.c_str());
    Serial.print("Connecting to WiFi ..");
    while (WiFi.status() != WL_CONNECTED) {
      Serial.print('.');
      delay(1000);
    }
    Serial.println(WiFi.localIP());
  }
}

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

How the Code Works
Let's take a quick look at the relevant parts of the code for this example.
Open the credentials namespace:

```
preferences.begin("credentials", false);
```

Get the SSID and password values using the getString() method. You need to use the key name that you used to save the variables, in this case, ssid and password keys:

```
ssid = preferences.getString("ssid", "");
password = preferences.getString("password", "");
```

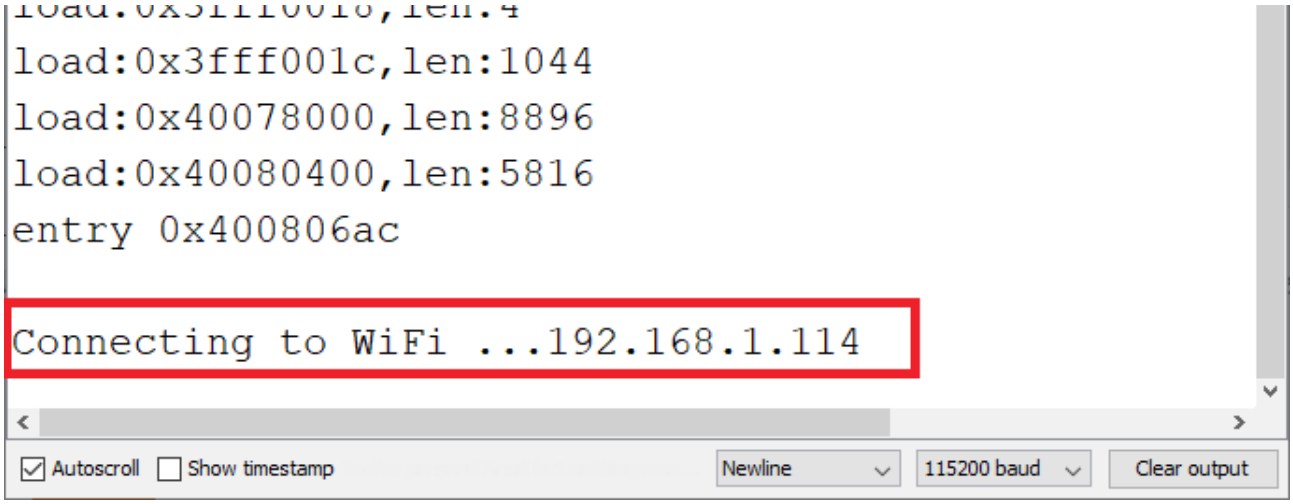
As a second argument to the getString() function, we passed an empty String. This is the returned value in case there aren't ssid or password keys saved on preferences.
If that's the case, we print a message indicating that there aren't any saved values:

```
if (ssid == "" || password == ""){
  Serial.println("No values saved for ssid or password");
}
```

Otherwise, we connect to Wi-Fi using the SSID and password saved on preferences.

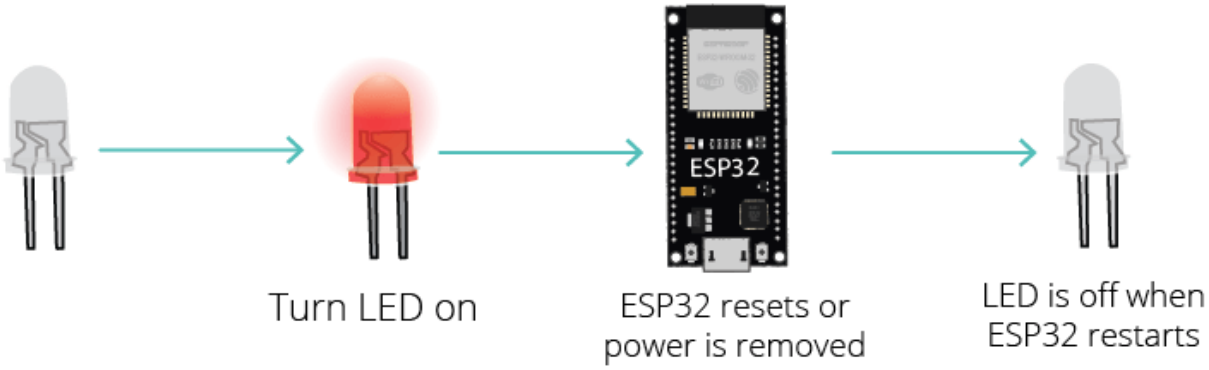
```
else {
  // Connect to Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid.c_str(), password.c_str());
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}
```

Upload this code to your board after the previous one (to ensure that you have the credentials saved). If everything goes as expected, this is what you should get on your Serial Monitor.

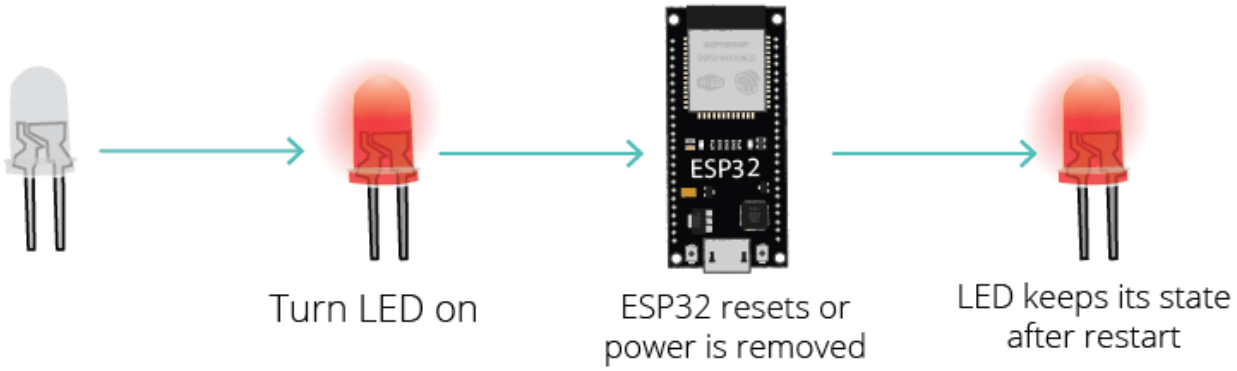


Example - Remember Last GPIO State After RESET
Another application of the Preferences.h library is to save the last state of an output. For example, imagine the following scenario:

1. You're controlling an output with the ESP32;
2. You set your output to turn on;
3. The ESP32 suddenly loses power;
4. When the power comes back on, the output stays off – because it didn't keep its last state.



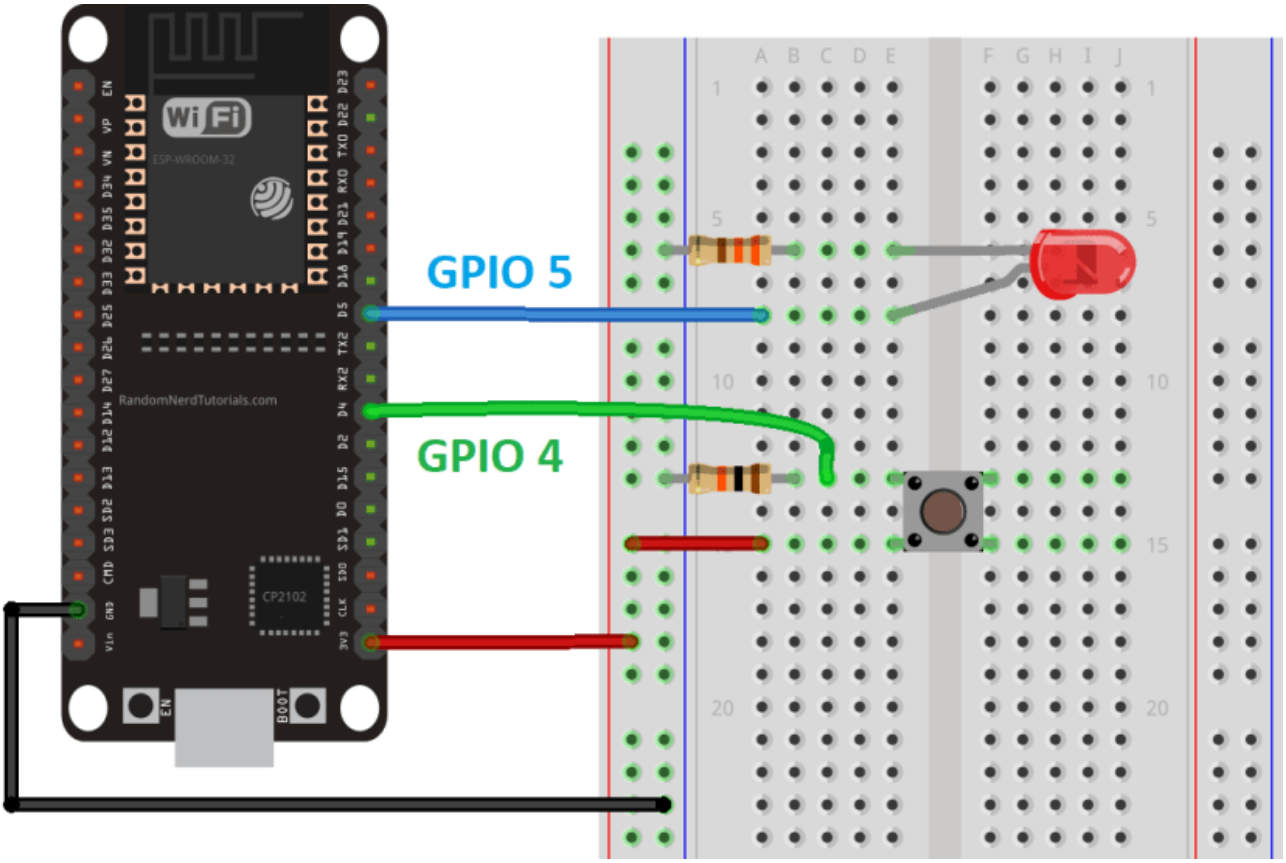
You don't want this to happen. You want the ESP32 to remember what was happening before losing power and return to the last state.
To solve this problem, you can save the lamp's state in the flash memory. Then, you need to add a condition at the beginning of your sketch to check the last lamp state and turn the lamp on or off accordingly. The following figure shows what we're going to do:



We'll show you an example using an LED and a pushbutton. The pushbutton controls the LED state. The LED keeps its state between resets. This means that if the LED is lit when you remove power, it will be lit when it gets powered again.

Schematic Diagram

Wire a pushbutton and an LED to the ESP32 as shown in the following schematic diagram.



Code

This is a debounce code that changes the LED state every time you press the pushbutton. But there's something special about this code – it remembers the last LED state, even after resetting or removing power from the ESP32. This is possible because we save the led state on Preferences whenever it changes.

```
#include <Preferences.h>

Preferences preferences;

const int buttonPin = 4;
const int ledPin = 5;

bool ledState;
bool buttonState;
int lastButtonState = LOW;

unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers

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

  //Create a namespace called "gpio"
  preferences.begin("gpio", false);

  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);

  // read the last LED state from flash memory
  ledState = preferences.getBool("state", false);
  Serial.printf("LED state before reset: %d \n", ledState);
  // set the LED to the last stored state
  digitalWrite(ledPin, ledState);
}

void loop() {
  int reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }
  lastButtonState = reading;
  if (digitalRead(ledPin)!= ledState) {
    Serial.println("State changed");
    // change the LED state
    digitalWrite(ledPin, ledState);

    // save the LED state in flash memory
    preferences.putBool("state", ledState);

    Serial.printf("State saved: %d \n", ledState);
  }
}
```

How the Code Works

Let's take a quick look at the relevant parts of code for this example.

In the `setup()`, start by creating a section in the flash memory to save the GPIO state. In this example, we've called it gpio.

```
preferences.begin("gpio", false);
```

Get the GPIO state saved on Preferences on the state key. It is a boolean variable, so use the getBool() function. If there isn't any state key yet (which happens when the ESP32 first runs), return false (the LED will be off)

.

```
ledState = preferences.getBool("state", false);
```

Print the state and set the LED to the right state:

```
Serial.printf("LED state before reset: %d \n", ledState);
// set the LED to the last stored state
digitalWrite(ledPin, ledState);
```

Finally, in the loop() update the state key on Preferences whenever there's a change.

```
// save the LED state in flash memory
preferences.putBool("state", ledState);

Serial.printf("State saved: %d \n", ledState);
```

Testing

Upload the code to your board and wire the circuit. Open the Serial Monitor at a baud rate of 115200 and press the on-board RST button.

Press the pushbutton to change the LED state and then remove power or press the RST button.

When the ESP32 restarts, it will read the last state saved on Preferences and set the LED to that state. It also prints a message on the Serial Monitor whenever there's a change on the GPIO state.