Using the ESP32 Preferences library

Mercury V1 manual · v1.0

https://d21c22pth15itr.cloudfront.net/support/mercuryv1/code-preferences/

The NVS / Preferences library lets you store settings and snippets of information in the NVS partition on the flash memory. This means these settings can be set and read without being lost when restarted or if power is lost. You can store most variable types in both signed and unsigned versions with this library.

In this example we make the reset button work as a on/off button for the led. You could also add deep sleep if it's in off mode to save power but there is more about that later. Each time you press the power button the Neopixel will either come on, or off depending on the toggle state.

The Preferences library supports all common variable types. For integers use prefs.putInt("key", 42) and prefs.getInt("key", 0) where the second parameter is the default if the key doesn't exist. For unsigned integers there's putUInt / getUInt, useful for counters like prefs.putUInt("boots", bootCount). Strings are stored with prefs.putString("wifi_ssid", "MyNetwork") and retrieved with prefs.getString("wifi_ssid", ""). For floating point values use prefs.putFloat("calibration", 1.0523) and prefs.getFloat("calibration", 1.0) — handy for sensor offsets. Boolean values use prefs.putBool("enabled", true) and prefs.getBool("enabled", false). To remove a single key call prefs.remove("key"), or to wipe an entire namespace use prefs.clear(). Each namespace is limited to 15 characters and keys to 15 characters. Always call prefs.end() when finished to free resources.

Using Arduino IDE? Our online programmer includes Mercury_Pins.h by default so the pin names work without issue. If you are using Arduino IDE or another programmer, copy the Mercury_Pins.h tab content and paste it into the top of your program.

```
/*
 * Mercury NVS Preferences Example
 * Toggles a value (0/1) in flash memory each time the board starts.
 * If 1: NeoPixel LEDs turn green. If 0: LEDs stay off.
 *
 * The value survives power cycles and reprogramming.
 * Reset by uploading new code that clears the namespace.
 *
 * No external libraries needed — Preferences and NeoPixel
 * are built into the ESP32 Arduino framework.
 */

#include "Preferences.h"
#include "Adafruit_NeoPixel.h"
#include "Mercury_Pins.h"

Preferences prefs;
Adafruit_NeoPixel pixels(4, LED, NEO_GRB + NEO_KHZ800);

void setup() {
  Serial.begin(115200);
  delay(500);

  // Power on NeoPixel
  pinMode(LEDPOWER, OUTPUT);
  digitalWrite(LEDPOWER, HIGH);
  delay(10);

  pixels.begin();
  pixels.clear();
```

```
    pixels.show();

    // Open NVS namespace ("demo" can be any name, up to 15 chars)
    prefs.begin("demo", false);  // false = read/write mode

    // Read the stored value (defaults to 0 if not yet saved)
    int state = prefs.getInt("toggle", 0);

    // Flip it
    int newState = (state == 0) ? 1 : 0;

    // Save the new value to flash
    prefs.putInt("toggle", newState);
    prefs.end();

    Serial.println("Previous state: " + String(state));
    Serial.println("New state: " + String(newState));

    if (newState == 1) {
        // All 4 pixels green
        for (int i = 0; i < 4; i++) {
            pixels.setPixelColor(i, pixels.Color(0, 30, 0));
        }
        pixels.show();
        Serial.println("LEDs: GREEN (reset to toggle off)");
    } else {
        Serial.println("LEDs: OFF (reset to toggle on)");
    }
}

void loop() {
    // Nothing to do — just demonstrating NVS storage
    delay(1000);
}

#pragma once
/*
 * Mercury (ESP32-C6) Pin Definitions
 * Board-specific GPIO assignments
 */

// ——— Status LED (NeoPixel) ———
#define LEDPOWER      3    // NeoPixel power (drive HIGH to enable)
#define LED           2    // NeoPixel data signal

// ——— I2C Bus ———
#define SDA           21   // I2C data
#define SCL           22   // I2C clock

// ——— Sensor Power ———
#define VACC          20   // Sensor power rail (drive HIGH to enable)

// ——— General Purpose Ports ———
#define GP06          6    // GP06 port
#define GP07          7    // GP07 port

// ——— High Current Output ———
#define OUT1          5    // High current output (e.g. pyro / relay)
```

```
// —— Battery Bar LEDs ——
#define BL1          4    // Battery LED 1 (lowest)
#define BL2          14   // Battery LED 2
#define BL3          15   // Battery LED 3
#define BL4          18   // Battery LED 4
#define BL5          19   // Battery LED 5 (highest)

// —— Indicators ——
#define DISK         8    // Disk activity LED

// —— Analogue / Detection ——
#define BATIN        0    // Battery voltage (1:1 divider)
#define USBDETECT    1    // USB power detect (HIGH = USB present)
#define BUTTON       9    // BUTTON on the board, boot button but can be used
```