START HERE

01

Configure by USB; verify by observation

In about **20 minutes**, load Wi-Fi settings, read a redacted status, restart the example application, then clear or recover its configuration. Saving a file is not the same as joining Wi-Fi.

Recognize your materials



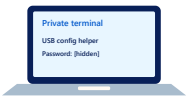
ESP32-S3-N16R8

Runs MicroPython and the lab app. Use native USB: left in the front view, antenna up.



USB data cable

Carries the file transfer. Keep other circuits and external power disconnected.



Computer + terminal

Runs the supplied helper. Use a private terminal for hidden input.



Permitted 2.4 GHz Wi-Fi

Use a WPA2-Personal lab network you may configure. Know its name and password.

1	2	3	4
Load	Verify	Restart	Clear / recover
Hidden input	Read safe status	Run the lab app	Remove lab config
USB → saved file	No password output	Try STA + IP	Keep your program

Saving settings and joining Wi-Fi are separate checkpoints.

USB transports configuration; the board later joins the Wi-Fi router as a station, or STA.

Prerequisites: working MicroPython from ML-C03, reliable USB from ML-C04, desktop Python 3.10+ and the example files. No prior network programming is required.

UNDERSTAND THE FILE

02

Separate private settings from code

A **configuration** is data that changes how a program behaves. This lab app reads one JSON file: `/lab_wifi.json`. JSON stores named fields such as a network name and password.

One lab file

/lab_wifi.json

schema: 1
ssid: your network name
password: hidden input

Validation before use

Exactly these three fields

SSID: 1–32 UTF-8 bytes

WPA2 passphrase: 8–63 printable ASCII characters

This is the course example's format. Stock MicroPython does not automatically interpret a file named lab_wifi.json.

SSID

The Wi-Fi network name. The limit counts UTF-8 bytes; plain ASCII characters use one byte each. Copy the exact name, including case.

Passphrase

This exercise accepts WPA2-Personal passwords of 8–63 printable ASCII characters. Enterprise, open networks and 64-digit hexadecimal keys are outside this lab.

Predict: if the password changes, do you need to reinstall MicroPython?
Check: no. Update the lab configuration and restart the app to apply it.

Keep settings private. The file on the board is plain text. Hidden typing and redacted output reduce accidental exposure; they do not encrypt board storage or replace control of physical USB access.

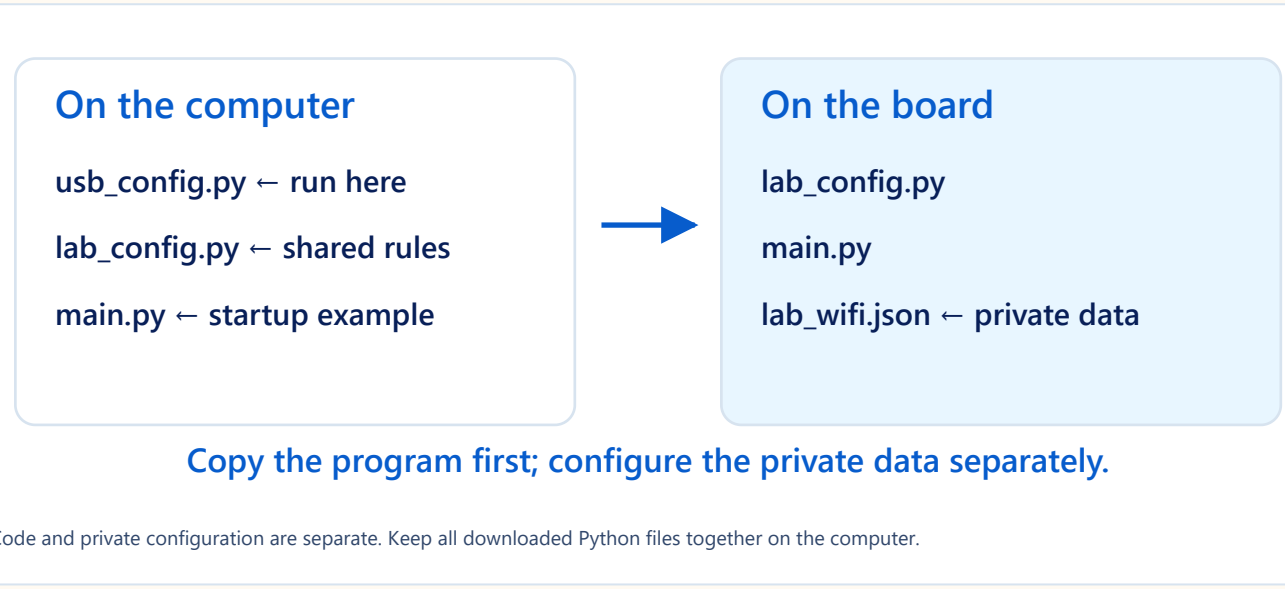
PREPARE

03

Prepare the files and computer tool

Save the four example files below in one folder named `usb-config-lab`. Keep their names unchanged. The helper runs on the computer; the application files are copied to the board.

usb_config.py lab_config.py main.py .gitignore



Code and private configuration are separate. Keep all downloaded Python files together on the computer.

Create a tool environment

Open a terminal in that folder. Confirm `python --version` before continuing.

```
python -m venv .venv
.venv\Scripts\python.exe -m pip install mpremote
```

PY below means `.venv\Scripts\python.exe` on Windows or `.venv/bin/python` on macOS / Linux. Replace PY with that executable path in every command.

> **Use your own working folder.** Do not run the lab in a shared project. The included ignore rules exclude lab Wi-Fi JSON files and backups from a new Git project.

CONNECT AND INSTALL

04

Install the small lab application

Use one USB data cable in native USB. In the component-side view, antenna up and sockets down, it is the **left socket**. Close other serial monitors and browser connections.

1 · List

List ports with the board unplugged.

2 · Connect

Plug into native USB and list again.

3 · Choose

Use the port that appears.

```
PY -m mpremote connect list
```

`PORT` below means your current runtime port, such as `COM7` on Windows. The example name is not a required port.

Save your existing project before replacing main.py

```
PY -m mpremote connect PORT fs cp :/main.py original-main.py
```

If there is no existing `main.py`, there is nothing to copy. Save other project files you need too. Then install the lab app:

```
PY usb_config.py --port PORT install --replace-main
PY usb_config.py --port PORT status
```

Example first status: `{"config_state":"missing","configured":false,"link_up":false}`.
The app files are installed; the private settings have not been supplied yet.

Know what changes: install replaces the lab's `main.py` and `lab_config.py`. It does not erase firmware or change the configuration file.

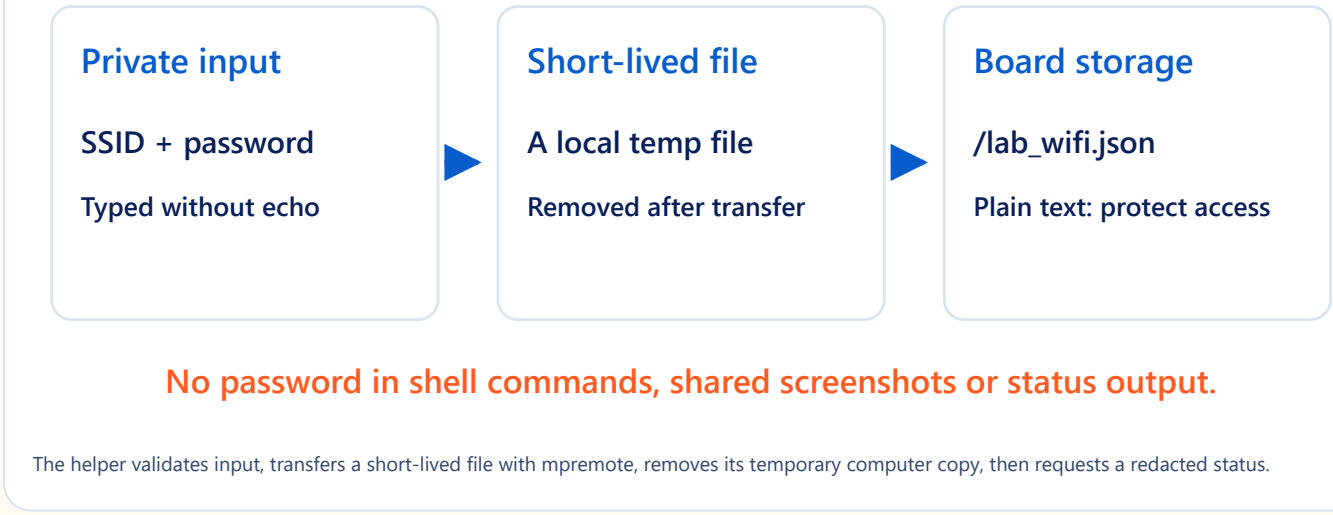
CONFIGURE

05

Load settings without showing them

Run the helper in a private terminal. It asks for both the SSID and password without showing the typed characters. Do not put real credentials into a shell command, screenshot or chat message.

```
PY usb_config.py --port PORT configure
```



The helper validates input, transfers a short-lived file with mpremote, removes its temporary computer copy, then requests a redacted status.

Example reply:

```
{"config_state":"saved","configured":true,"link_up":false}
Saved only. Restart the lab app to test these settings.
```

The helper checks the schema and field lengths before copying. If hidden input is unavailable, it stops instead of falling back to visibly typed input.

A file transfer can be interrupted. Keep USB connected while the command runs. Back up existing settings privately before replacing them. If the resulting file is invalid, use the recovery sequence on page 9.

Check: the status should contain no SSID or password. A saved file alone has not yet proved a Wi-Fi connection.

VERIFY THE RIGHT THING

06

Read status without confusing the two checks

The helper's **status** action reads the file and the current station link. It does not apply a new file. An older link may still be active until the app restarts.

```
PY usb_config.py --port PORT status
```

File checkpoint

config_state: saved

configured: true

The fields are valid and stored.

Radio checkpoint

link_up: false / true

Current STA connection only

Restart to apply the saved file.

An old Wi-Fi link can exist before a new file is applied.

File validity and radio connection answer different questions.

config_state

`missing`: no lab file.
`invalid`: data fails parsing or validation.
`saved`: valid data is present.
`unreadable`: the app could not read it.

link_up

The current STA interface reports a connection with an IP address. It does not say that a newly copied file was used for that connection.

Predict: a new file is saved while an old Wi-Fi link remains up. Does `link_up:true` prove the new password worked?
Check: no. Restart the app and observe a fresh result.

APPLY AND PROVE

07

Restart and observe a fresh connection

Open the board console with the **normal runtime port**. Press Ctrl-C for >>>, then Ctrl-D at that normal prompt to restart Python and run the installed `main.py`.

```
PY -m mpremote connect PORT repl
```

Normal REPL	Fresh startup	Observe
Ctrl-C → >>>	Read lab_wifi.json	wifi_start state
Then Ctrl-D	Connect with a timeout	connected / timeout

A new “connected” startup result requires STA association and an IP.

Use Ctrl-D at the normal prompt. A raw tool operation can soft-reset without running main.py.

Look for a new line after the restart:
`{"event":"wifi_start","state":"connected"}`
In this app, that result follows a fresh connection attempt with the saved file and a station IP address.

The attempt is limited to about 15 seconds. A `timeout` or `connect_failed` result means investigate the network and settings; it does not mean the firmware must be erased.

Persistence check: unplug USB, reconnect, rediscover the port if needed and open the console again. Use Ctrl-C then Ctrl-D to observe another fresh startup. The saved file should still be available.

Ctrl-] exits mpremote before another tool uses the port. A router connection is not proof of Internet or cloud-service access.

CLEAR DELIBERATELY

08

Clear the lab configuration, keep the program

Make a private backup if you need the existing settings. The helper refuses to overwrite an existing backup name. The backup contains credentials: keep it private and out of Git.

```
PY usb_config.py --port PORT backup --to lab_wifi.backup.json
PY usb_config.py --port PORT clear
```

Clear this lab file

/lab_wifi.json

Disconnect lab STA Wi-Fi

Next startup: missing

Keep these

main.py + lab_config.py

Other project files

Private backups you made

Clear is a named-file operation. It does not erase the chip or delete your application files.

Expected redacted state: `config_state:"missing"`, `configured:false`, `link_up:false`.
After a normal restart, the lab app reports `wifi_start` with state `missing`.

Verify: restart and check for `missing`. You can configure again using the same program; there is no need to reinstall MicroPython.

Clear is not secure erasure. Local backups remain, and deleting a file is not a guarantee that old flash contents are unrecoverable. Use the device's documented reset process before transferring ownership.

RECOVER WITHOUT A FULL ERASE

Recover a malformed lab file

09

A damaged configuration is different from damaged firmware. The app leaves an invalid file in place so you can preserve it privately before replacing it.

1 · Identify

invalid, not missing

Do not print the file

2 · Preserve

Private backup

Use a fresh filename

3 · Replace

Clear → configure

Restart and verify

A malformed lab file usually needs file recovery, not a firmware erase.

Use a fresh private backup name; do not print or share the contents.

Optional practice with a deliberately broken example

First back up any settings you need. Save invalid_lab_wifi.txt beside the Python files. This next command replaces only the lab config with a broken, non-sensitive example:

```
PY -m mpremote connect PORT fs cp invalid_lab_wifi.txt :/lab_wifi.json
PY usb_config.py --port PORT status
```

Expect `config_state:"invalid"` and `configured:false`. Then preserve, clear and enter valid settings:

```
PY usb_config.py --port PORT backup --to lab_wifi.bad.backup.json
PY usb_config.py --port PORT clear
PY usb_config.py --port PORT configure
```

Finish the recovery: repeat page 7's normal restart and connection observation. Use ML-C10 only if the runtime itself cannot be recovered through the normal USB/REPL route.

TROUBLESHOOT AND COMPLETE

10

Use the first missing observation

Change one thing and repeat the earliest failed checkpoint. Preserve useful facts; leave SSIDs and passwords out of diagnostic captures.

No port	Check USB and the current port.
Busy port	Close other serial owners.
Saved, no Wi-Fi	Restart; check network + password.
Invalid file	Back up privately; replace this file.

Start at the USB link before blaming the Wi-Fi settings or firmware.

Saved, but timeout

Confirm the permitted 2.4 GHz WPA2-Personal network is available. Re-enter the exact settings privately and restart. A wrong password can pass the format checks.

Recovery still fails

Check the current port, close competing tools and confirm the lab app files are present. A missing or malformed lab file normally needs a file fix.

Your completion check

- [ ] Load settings with no credential literal in the shell history.
- [ ] Read a status that contains no SSID or password.
- [ ] Observe a fresh `connected` result after normal startup.
- [ ] Clear only the lab file and confirm `missing` after restart.
- [ ] Explain how to preserve and replace an invalid lab file.

> **Explain the distinction:** `configured:true` means valid settings are stored. A fresh app startup result of `connected` means the current saved settings completed the lab's station connection check.