

# PYTHONPATCHER.PY

## INSTRUCTIONS TO PATCH AND UPLOAD ESP BINARIES

### FOR USERS: PATCHING THE BINARY WITH YOUR CREDENTIALS

#### DOWNLOAD

Download the provided:

- \*.bin executables,
- esptool.py,
- espota.py,
- PythonPatcher.py or PythonPatcherTE.py

to any empty directory of your choice.

#### ON WINDOWS

Test if Python is already installed:

- Start the Windows Command line (Windows-Key, cmd)
- Type `cd [your chosen directory]`, return
- Type `python3`, return.
  - If you get an error that Python is not installed, get the free Python3 from the Windows store (on Windows11, you will directly be offered to install.
  - If you get a Python greeting ending with `>>>`, Python is installed
  - Leave Python with `exit()`, return

#### ON MACOS OR LINUX

- Start the Terminal
- Python3 should be already installed

#### FOR USERS OF ALL OPERATING SYSTEMS

- Load the ESP flasher modules, if they were not already:  
`pip3 install esptool`      *ESPTool.py is the ESP serial flasher*  
`pip3 install espota`      *ESPOTA.py is the flasher "Over The Air"*
- Load the following modules used by PythonPatcher, if they were not already:  
`pip3 install geocoder`      *geocoder gets your geo coordinates*
- Your System is now ready to use PythonPatcher and flash your devices.  
*N.B. If you never uploaded to ESP devices before, you may need to install the drivers corresponding to the Serial-USB chip installed on your board: cf. Appendix.*

## COMMON PART FOR ALL OPERATING SYSTEMS

### START THE PYTHON PATCHER

- Type `python3 PythonPatcher.py`, return.
- If you intend to use a device

```
Welcome to PythonPatcher for ESP devices
from RIN67630 @ https://github.com/rin67630/ESP_Binary_patcher
Please select a file from:
[1] ESP_SwissArmyKnife_NoScreenRev07_2024.bin
[2] ESP_SwissArmyKnife_OLED128x64Rev07_2024.bin
Select .bin file[1-%s]:
```

Enter the number of the chosen file to upload.

*N.B: a filename containing “\_update” or “\_patched” will not be patched, but be immediately proposed to (re) -upload Over-The-Air.*

```
Working on ESP_SwissArmyKnife_NoScreenRev07_2024..bin, let's begin to patch !
Enter SSID:
Enter Password:
Enter Device Name:
Enter Cloud User Name:
Enter Device Credentials:
```

- Enter the credentials as requested.  
*If you enter nothing, the defaults defined in PythonPatcher will be used instead.*  
Then, PythonPatcher will determine your geographical data from your IP:

```
Your IP appears to be: 90.101.106.251
Accordingly, you appear to be in: Sarrebourg
Time zone offset is: 3600 seconds ahead of Greenwich
Daylight saving offset is: 7200 seconds ahead of Greenwich
Your latitude is: 48.7356
Your longitude is: 7.0572
Patch the ESP device with these values?(y), or manually enter values?(n)
```

*If you will be using your ESP device where you are currently, enter “y”,*

*If you will be using that device elsewhere, enter “n” and enter the data alike:*

```
Please enter GMT offset in seconds ahead, or behind (-) Greenwich:3600
Please enter Daylight Saving offset in seconds ahead, or behind (-) Greenwich: 7200
Please enter the device's latitude (-180...180): 45.577
Please enter the device's longitude (-90...90): 6.5757
```

- Enter the credentials as requested.  
*If you enter nothing, the defaults defined in PythonPatcher will be used instead.*
- At the end the, #####\_patched.bin file will be written to the desktop.

```
Patched binary ESP_SwissArmyKnife_NoScreenRev07_2024._patched.bin saved
```

- (if you made a mistake, just redo `python3 PythonPatcher.py`, return).

## UPLOADING THE PATCHED BINARY TO YOUR ESP DEVICE

PythonPatcher continues with the flashing process:

Flashing it now to the ESP device on the first valid serial port?

- If it is the initial flashing, answer Y
- For subsequent flashings, you may flash over serial (answer Y) or flash Over-The Air without need to plug your device to the computer (answer N)

```
C:\Users\MiFi\Desktop>python3 esptool.py write_flash -z 0x0000 #####_patched.bin
esptool.py v3.3
Found 1 serial ports
Serial port COM7
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting...
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: bc:dd:c2:7b:14:76
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash will be erased from 0x00001000 to 0x0007ffff...
Compressed 517200 bytes to 375363...
Wrote 517200 bytes (375363 compressed) at 0x00001000 in 33.2 seconds (effective
124.5 kbit/s)...
Hash of data verified.
```

- Now that you have initially flashed your device, answer N to the OTA proposal.
- The PythonFlasher ends, your device is online and ready !
- You will need a last step: take notice of your device's IP Address !  
Type `ping [your device name]`

```
Pinging Witty.fritz.box [192.168.188.73] with 32 bytes of data:
Reply from 192.168.188.73: bytes=32 time=34ms TTL=255
Reply from 192.168.188.73: bytes=32 time=32ms TTL=255
Reply from 192.168.188.73: bytes=32 time=31ms TTL=255
Reply from 192.168.188.73: bytes=32 time=15ms TTL=255

Ping statistics for 192.168.188.73:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 15ms, Maximum = 34ms, Average = 28ms
```

- Take note of the IP address of your ESP, you will need it frequently.

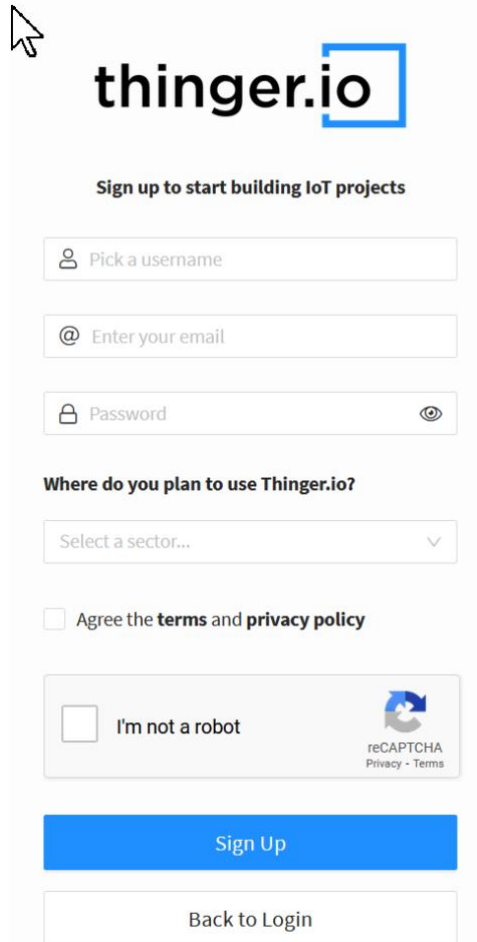
## FOR USERS OF THINGER.IO CLOUD SERVICE

### PREPARING YOUR DEVICE AT THINGER.IO

#### CREATING A FREE ACCOUNT

Start with <https://console.thinger.io/signup>

Enter your Data:



thinger.io

Sign up to start building IoT projects

Pick a username


@ Enter your email

Password ☐

Where do you plan to use Thingier.io?

Select a sector... ▾

☐ Agree the **terms** and **privacy policy**

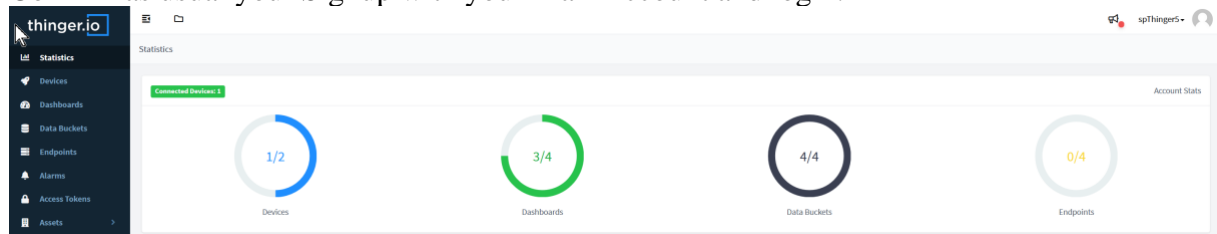
☐ I'm not a robot  reCAPTCHA  
Privacy - Terms

Sign Up

Back to Login

Your username will be the credential **Cloud User Data**

Confirm as usual your Signup with your Mail Account and login.



*With your free account you can use 2 devices, 4 dashboards, 4 data buckets, 4 endpoints*

#### CREATE A DEVICE

Click on Device and on Add Device:

### Device Details

Device Configuration

Device Type ⓘ

IOTMP Device (Thingier.io protocol)

Device Id ⓘ

Enter device identifier

Device Credentials ⓘ

Enter device credentials

Device Information

Device Name ⓘ

Optional device name

Device Description ⓘ

Optional device description

Advanced Options

Asset Type ⓘ

☐

Select Type...

Asset Group ⓘ

☐

Select Group...

Product ⓘ

☐

Select Product...

Enabled ⓘ

☒

Enter a Device Id (up to 16 ASCII alphanumeric chars, no space)

This will be your **Device Name**.

Enter a password for your device (you may use the generator button)

This will be your **Device Credentials**

You may enter an optional Device name and a Device description (just used on Dashboards and Reports)

Your device is now ready to be used and you have defined all credentials to flash it.

## FOR USERS OF THINGER.IO CLOUD SERVICE WITH APPLICATIONS FROM RIN67630

Several of my Applications at Github are using thinger.io online facilities and dashboard.

- [https://github.com/rin67630/ESP\\_Karajan](https://github.com/rin67630/ESP_Karajan) (my generic framework for the other applications)
- <https://github.com/rin67630/Sound-pressure-level-meter-Booster>
- [https://github.com/rin67630/Victron\\_VE\\_on\\_Steroids](https://github.com/rin67630/Victron_VE_on_Steroids)
- <https://github.com/rin67630/Drok-Juntek-on-steroids>

## PREPARING THE BUCKETS AT THINGER.IO

Thingier's Buckets are recording long term information. They can be used for time series trending, the last value can be displayed as values in a dashboard, and you can download their content as CSV for processing in a Spreadsheet.

My applications generally use the same bucket structure:

- Bucket EVENTS. (recording all recognized events, like threshold exceeding, alarms ...)
- Bucket MIN. (recording main values at a minute pace)
- Bucket HOUR (recording values at a hourly pace, including weather information)
- Bucket DAY (recording daily summaries, including arrays of hourly values)

Please create all these buckets accordingly to ensure proper dashboard operations:

Creating a bucket:

The screenshot shows the 'Add Bucket' interface in the Thingier.io dashboard. The left sidebar contains navigation links for Statistics, Devices, Dashboards, Data Buckets (selected), Endpoints, Alarms, Access Tokens, Assets, File Storages, Products, Projects, Plugins, Toolbox, and Administration (User Accounts, Cluster Hosts, Domains). The main content area is titled 'Buckets > Add' and 'Bucket Details'. It features a 'Bucket Settings' section with 'Bucket Id' set to 'MIN'. The 'Bucket Information' section has 'Bucket Name' and 'Bucket Description' fields with placeholder text. The 'Bucket Configuration' section shows 'Enabled' as a toggle switch and 'Data Source' as 'From Device Write Call'. The 'Advanced Options' section includes dropdowns for 'Asset Type', 'Asset Group', and 'Product'. A blue 'Add Bucket' button is located at the bottom left of the form.

The Bucket Id (EVENTS, MIN, HOUR, DAY) and the Data Source “From Device Write Call) are mandatory.

The Bucket Name and Bucket Description are free: enter any custom text.

## PREPARING THE DASHBOARDS AT THINGER.IO

Most of my applications are distributed with Dashboard Templates.  
Files like:

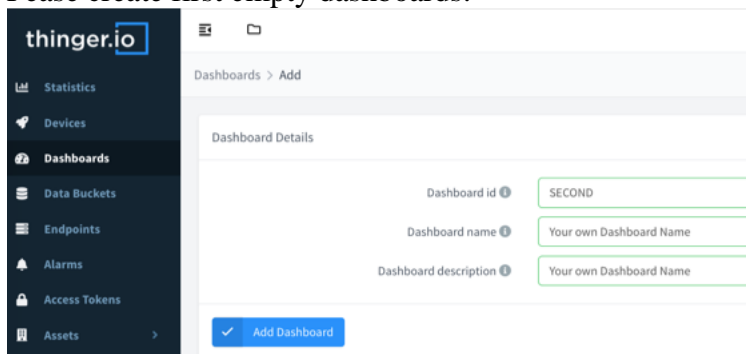
- [t1 Frontpage Thinger.h](#)
- [t2 Statistics Thinger.h](#)
- [z Thinger.Min.h](#)
- [z Thinger.Sec.h](#)
- ...

Contain all settings of extensive dashboards, that took hours to configure.

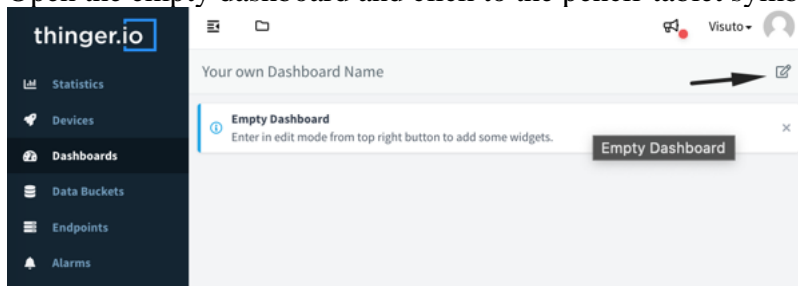
Thes templates in my precompiled distributions contain placeholders, which will be patched by PythonPatcherTE. After having been patched with your data these files get the postfix `_patched`.

You can use these templates to immediately start with gorgeous dashboards, which you may tailor to your taste later.

Pease create first empty dashboards:



The Dashboard Name and Dashboard Description are free: enter any custom text. Save.  
Open the empty dashboard and click to the pencil-tablet symbol at the right:



Then click to Settings and then to Developer.

Leave the windows open and open the patched template file in a text editor.

Click anywhere on the code and then type [ctrl-A] to select all the text. ( [cmd-A] on a Mac.).

The click [ctrl-C] to copy all the text. ( [cmd-C] on a Mac.).

Return to the Dashboard Developer view.

Click anywhere on the code and then type [ctrl-A] to select all the text. ( [cmd-A] on a Mac.).

The click [ctrl-V] to paste all the text. ( [cmd-V] on a Mac.).

Save.

Enjoy your new dashboard.

## FOR APPLICATION PROGRAMMERS: DESIGN RULES

### CREDENTIALS AND GEO-INFORMATION

Writing ESP applications for PythonPatcher is extremely easy.

There is practically no code required on the ESP device.

No extra code to initialize, no need to save to EEPROM, which remains fully available,  
You just define some compiler macros exactly as described (with all the spaces):

```
#define DEVICE_NAME    "DEVNAME"    // Device name    (exactly 16 chars incl spaces)
#define WIFI_SSID      "WIFISSID"    // Wifi name    (exactly 16 chars incl spaces)
#define WIFI_PASS      "WIFIPASS"    // Wifi passwd    (exactly 24 chars incl spaces)
#define CLOUD_USERNAME "CLOUDNAM"    // Cloud User name    (exactly 16 chars incl spaces)
#define DEVICE_CREDENTIALS "DEVCCRED" // Devices credentials    (exactly 16 chars incl spaces)

#define TZ_OFF          "TZ_OFF "    // Offset to GMT in secs    (exactly 8 chars incl spaces)
#define DST_OFF          "DST_OFF "    // Summer Offset in secs    (exactly 8 chars incl spaces)
#define LONGITUDE        "LONGTD "    // Longitude    (exactly 8 chars incl spaces)
#define LATITUDE         "LATITD "    // Latitude    (exactly 8 chars incl spaces)
```

These definitions are identically defined in PythonPatcher and will be replaced by user's own values in the binary, extended with nulls to match the initial length.

The five first parameters are usually needed to connect to Wi-Fi and to a cloud service.

In your code you just use the macro-names:

```
WiFi.begin(WIFI_SSID, WIFI_PASS);
```

```
WiFi.hostname(DEVICE_NAME);
```

```
ArduinoOTA.setHostname(DEVICE_NAME);
```

```
ThingierESP8266 thing(CLOUD_USERNAME, DEVICE_NAME, DEVICE_CREDENTIALS);
```

The time information and geo coordinates are numeric, they are however patched as String() to abstract processor specific differences like endianness and the CPU's bit width.

You must convert them to numeric at the beginning of your global variables' definition:

```
int tz_off = int(TZ_OFF);
int dst_off = int(DST_OFF);
float longtd = float(LONGTD);
float latitd = float(LATITD);
```

Then you can use the numeric variables regularly in your code.

Here an example on how to initialize NTP:

```
void getNTP() {configTime(tz_off,dst_off,"pool.ntp.org");}
```

Here an example of how to call OpenWeatherMaps:

At the beginning of your global variables' definition

n.b. *This is my private key for initial testing!*

*Please don't abuse: no more than 1 call per 10 minutes in average.*

*May be recalled any time, use your own free key on production !*



```
#define OPEN_WEATHER_MAP_APP_ID "7f02173c19eed016c4797f4ca4251fcc"
```

In Setup() *Compute once the URL to call OpenWeatherMaps*

```
sprintf(openWeatherMapsURL, "https://api.openweathermap.org/data/2.5/weather?lat=%07.4f&lon=%07.4f&appid= OPEN_WEATHER_MAP_APP_ID", latitd,  
longtd);
```

*n.b.: This is one line, don't be fooled by word's wrapping !*

In Loop() *This is not all code required, just the part to retrieve the info, look in my projects to find all code to retrieve weather info from OpenWeatherMaps.*

```
...  
http.begin(wifiClient, openWeatherMapsURL);  
int httpCode2 = http.GET();  
if (httpCode2 == HTTP_CODE_OK) JSONpayload = http.getString();  
...
```

That's it.

You don't need more to get your Wi-Fi and your Cloud service on-line, the time synchronized and the weather information available for your device.

## APPENDIX

Install additional USB-Serial drivers on Windows:

### SILABS

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

### PROLIFIC

<https://oemdrivers.com/usb-prolific-pl2303>