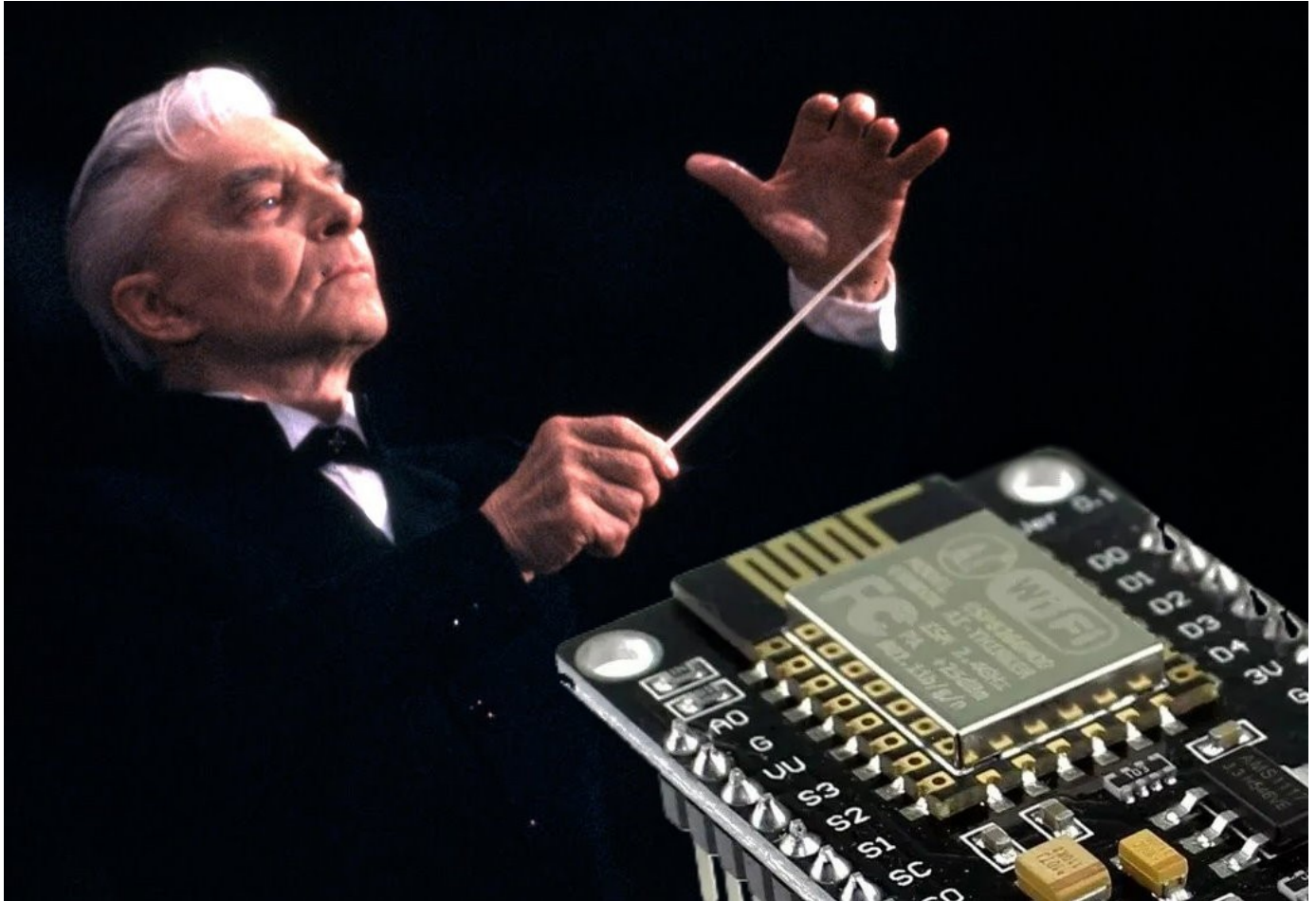


ESP KARAJAN

MY OWN WAY TO PROGRAM ESP DEVICES

... to do a lot of things seamlessly and to communicate together.



Herbert von Karajan was a brilliant conductor, knowing how to get the uttermost from the musicians of his symphonic orchestra, triggering everyone exactly on time in a gorgeous harmony. I am proud to – immodestly– borrow his name to my project.

Espressif's development boards **ESP8266** and **ESP32** are versatile and powerful and very low-cost micro-controllers. They can be programmed with the no-frills Arduino IDE, which is very accessible to beginners. The backside of the medal, is that you have to program „bare metal“ and must take care of almost everything yourself.

The chip is also single threaded for the user, that means, you can't do several things in parallel. So, how will you manage to program a complex data acquisition, with several different timings, while updating a display, providing a user menu, exchanging data between two ESP devices, transmitting reports to a remote PC?

That is where ESP-Karajan comes into the game...

TABLE OF CONTENTS

Overview.....	3
What is ESP-Karajan?.....	3
Scheduler.....	3
Communication.....	3
Menu.....	4
Dashboards.....	5
Non-Volatile Memory.....	5
Inter-ESP Communication (option).....	5
Computer Communication (option).....	5
The Arduino IDE: Tabs.....	6
What is ESP-Karajan not?.....	6
Configuration.....	7
Config.h.....	7
Credentials.h.....	8
a_Libs_Vars.....	8
b1_Karajan_Functions.....	9
b2_User_Functions.....	9
c_Setup.....	9
D_Menu.....	10
e1_Instant_Data.....	10
e2_Fast_Data.....	10
e2_Slow_Data.....	10
f_Stats.....	10
G_Display.....	10
H_Serial.....	12
I_Wireless.....	12
S_Scheduler.....	12
t1_Frontpage_Thingier.h t2_Statistics_Thingier.h.....	12
Thingier.io dashboards.....	13
Create an account.....	13
Setup a device.....	13
Setup resources and properties.....	13
Buckets.....	13
Dashboards.....	14
Preconfigured dashboards.....	15
Conclusion.....	17
Which devices are supported?.....	17
Beginner's Guide.....	18
Installing the IDE.....	18
Configure the IDE.....	18
Install the boards and libraries on the Arduino IDE.....	18
Boards:.....	18
Global Libraries.....	18
Project's libraries.....	18
Get the Arduino IDE Files from GitHub.....	19
Configure the software.....	19
Compile your sketch.....	19

OVERVIEW

WHAT IS ESP-KARAJAN?

ESP-Karajan is a kind of operating system for the ESP devices, taking away the time consuming and unproductive tasks of initializing the processor, WiFi connection, the displays, online dashboards, the measurement chips, the communication between ESP devices and the connection to a remote PC.

SCHEDULER

Karajan contains a collaborative scheduler written for ease of use and clarity easily accessible to non-programmers without IT gibberish.

It provides several levels of runtime processing

- Instant data (highest priority, no fixed timing, for asynchronous communication with other devices)
- Fast data (runs every 125ms)
- Menu (runs every 125ms)
- Slow data (runs every second)
- Statistics (runs every second)
- Display (runs every second)
- Serial reports (runs every second)
- Wireless communication (runs every second)

The scheduler distributes the tasks running every second evenly over the whole second.

Additionally timing flags are provided:

- NewMinute
- Minute Expiring
- NewHour
- HourExpiring
- NewDay
- DayExpiring

which allow to process statistics at the right time.

These timing flags rely on the time given from a NTP server over the WiFi connection.

COMMUNICATION

Upon booting the ESP module communicates over its serial port.

You may choose to keep it that way, or to switch to telnet which makes the ESP remotely controllable and provides a galvanic separation between the ESP and the controlling PC.

Another advantage of Telnet is to free the UART of the ESP for a process usage.

Alternatively, you can redirect the interaction to a software serial port if you prefer remaining local and wired.

MENU

Karajan provides a menu to control displays, start reports, change setpoints, and start actions. Each action is started from a single character:

- V. display the regular device values every second

Values Report

```
BatV:13.70 BatI:00.08 BatW: 01.05 PanV: 01.14 PanI: 00.99 PanW: 01.12 LoadI: 00.69 LoadW: 09.45 BatR: 0.0154 POC:91.3
BatV:13.70 BatI:00.08 BatW: 01.05 PanV: 01.14 PanI: 00.99 PanW: 01.12 LoadI: 00.69 LoadW: 09.45 BatR: 0.0154 POC:91.3
BatV:13.70 BatI:00.08 BatW: 01.05 PanV: 01.14 PanI: 00.99 PanW: 01.12 LoadI: 00.69 LoadW: 09.45 BatR: 0.0154 POC:91.3
BatV:13.70 BatI:00.08 BatW: 01.05 PanV: 01.14 PanI: 00.99 PanW: 01.12 LoadI: 00.69 LoadW: 09.45 BatR: 0.0154 POC:91.3
```

- D. display the debugging device values every second

Debug Report

```
A0: 21 BatV:13.70V BatI:0.077A CellV: 3.423V BatPoC:91.3% IOhm:0.0154R State: Off
A0: 21 BatV:13.70V BatI:0.077A CellV: 3.423V BatPoC:91.3% IOhm:0.0154R State: Off
A0: 21 BatV:13.70V BatI:0.077A CellV: 3.423V BatPoC:91.3% IOhm:0.0154R State: Off
```

- B. Battery Ah report

B Battery Stats

Hour	00	01	02	03	04	05	06	07	08	09	10	11
Bat Ah	-0.72	-0.72	-0.72	-0.72	-0.72	-0.73	-0.73	-0.62	00.57	02.64	02.92	06.17
Bat V	13.36	13.35	13.34	13.34	13.33	13.31	13.31	13.30	13.27	13.29	13.28	13.31
Hour	12	13	14	15	16	17	18	19	20	21	22	23
Bat Ah	05.98	07.89	10.12	10.31	05.82	00.76	01.03	-0.20	-0.68	-0.71	00.00	-0.72
Bat V	13.31	13.41	13.37	13.45	13.60	13.60	13.69	13.69	13.69	13.37	00.00	13.36

- b. (lower case) Resets the integration values (here Ah)
- J. (upper case) Display the job execution time monitoring every second
This shows the current execution time in mS, for each periodical job 100ms being the absolute maximum bearable. The second column displays the maximum until reset.

Job Timing

Job Durations(mS) Current - Max

```
Sche:001 - 001
Fast:000 - 004
Slow:007 - 012
Stat:000 - 298
Disp:000 - 001
Seri:-1000 - 004
Wifi:001 - 002
```

- j. (lower case) resets the job execution time monitoring.
- W displays the weather list

Weather list :

Temp: 6.1°C Hum: 77.0% Press: 1009mBAR WindSpeed: 8.8m/s Direction: 310°N Clouds: 75% Summary: broken clouds

- 0, 1,2,3,4,5,6 switch display pages (0 = stop display)
- +. -. Within a display page, a corresponding setpoint can be changed with + or -
- Z. resets the device: it will reboot. This cuts the Telnet transaction.
- X. resets the device and additionally erases the WiFi credentials.
- Q Quits the Telnet session

DASHBOARDS

ESP Karajan loves Thinger.io ! cf: <https://thinger.io/>

With this great dashboard provider, you can configure gorgeous dashboards online without the hassle of MQTT as a man-in-the-middle and fiddling with Influx databases and Grafana Servers.

You communicate directly with your dashboards, which can be edited and configured in-place.

Read more at the Chapter [Thinger.io dashboards](#)

NON-VOLATILE MEMORY

Karajan uses Thinger.io also as a provider of non-volatile memory required when long-term statistics are to be made.

Data is stored in “properties” and automatically read back upon restarting the ESP.

This has also invaluable advantages over EEPROM that

- you can replace a defective ESP without losing data.
- you can update the data as frequently as you want, without risk of wearing prematurely the low-end EEPROM of the ESP device that is designed for ~100 refreshes only.
- your data is also visible and modifiable within thinger.io

INTER-ESP COMMUNICATION (OPTION)

With Karajan, two ESP-Modules can communicate to each other.

Typically, one ESP module placed outdoors performs the data acquisition, sends the results to another ESP module indoors that displays the values, processes the reports and the statistics.

The communication occurs over UDP messages.

To save resources, the only security is the WiFi encryption, UDP messages send the raw variable’s structure content. There is no header, the only validation is the message length.

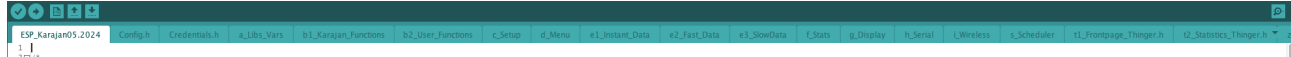
COMPUTER COMMUNICATION (OPTION)

Your ESP can also periodically send reports to e.g. a Raspberry Pi that will append the periodical protocol to a log file

THE ARDUINO IDE: TABS

The Arduino IDE provides a less used, but great feature: **Tabs**.

Instead of writing all the program in one huge single file, you can choose to split the code in separate functional subparts.



The code of Karajan uses these tabs:

- ESP_Karajan... (Only comments here)
- Config.h. (here you enter your options)
- Credentials.h (here you enter your secret credentials)
- a_Libs_Var (here the libraries are invoqued and the common Variables declared)
- b1_Karajan_Functions (here the (fixed) Functions of Karajan are defined)
- b2_User_Functions (here you define your own functions)
- c_Setup (the setup procedure)
- d_Menu (the menu content)
- e1_Instant_Data (your program data with the highest priority and no fixed periodicity)
- e2_Fast_Data (periodic data paced at 125mS)
- e3_Slow_Data (periodic data paced at 1S)
- f_Statistics (statistics executed after the periodic data, paced at 1S)
- g_Display (display processing, paced at 1S)
- h_Serial (reports to the user, paced at 1S)
- i_Wireless (reports to a computer, another ESP or thinger.io, paced at 1S)
- s_Scheduler (the main program:loop that controls all other tabs)

Additional tabs are not part of the program, but contain the structure of related dashboards on thinger.io They are placed here for convenience only

- t_Frontpage_Thinger.h
- t_Statistics_Thinger.h

Working with tabs is very convenient as many functions involve several modules and you can jump from one to the other module (tab) easily. The IDE even remembers where you were in every tab and brings you exactly ther back when you change tabs.

WHAT IS ESP-KARAJAN NOT?

Karajan is not a real-time operating system, it will not prevent one of your program parts to block everything if you don't program carefully. You must take care that every part of your program never takes longer than a few tens of mS to execute.

So longer delay() instructions and communication handshakes waiting for a response must be avoided.

CONFIGURATION

Karajan is full of options and can be tailored to your wishes.

The options can be configured before compiling at the tab Config.h:

CONFIG.H

```
// N.B. Compile sketch with following board settings:
//- for option 8266:      LolinWEMOSD1 (Clone)
//- for option TTGO:      TTGO T1
//- for option HELTEC LoRa: HELTEC WiFi Lora32 (Not V2 !) from HELTEC, not generic
//----- HARDWARE OPTIONS -----
#define SCREEN_IS_128x64 // _NONE , _64x48 , _128x64, _TTG0, _HELTEC, _WEMOS32 _HW364A ;
#define SCREEN_IS_REVERSED // _IS_NORMAL, _IS_REVERSED To turn the display 180° if required
#define BRIGHTNESS 128 // PWM value for default brightness with TTGO 0=totally dark;255=totally shiny
```

Here you define which screen you use (if any)

```
//----- SOFTWARE OPTIONS -----
#define DASHBRD_IS_NONE // _NONE , _THINGER // (Internet Dashboard)
#define TERM_IS_TELNET // _TELNET, _SERIAL , _SOFTSER, _2SERIAL // defines where do Menus and Reports occur: _SERIAL and D7_IS_VICTRON are mutually exclusive )
#define UDP_IS_NONE // _NONE , _SEND , _RECEIVE // (UDP Inter-ESP Communication)
#define ESP_UDP_ADDR "192.168.188.85" // (IP of the receiving ESP having UDP_IS_RECEIVE)
#define ESP_UDP_PORT 4200 // (Port used to send/receive Values to other ESP)
#define COM_IS_NONE // _NONE , _HOURLY // Periodical reports to computer
#define COM_UDP_ADDR "192.168.188.96" // (IP of the receiving computer for night reports)
#define COM_UDP_PORT 4230 // (Port used to send/receive Values to other computer)
// #define // DEVICE_NAME "Karajan" // Name of the device used as Hostname and at Thinger.io
#define DEVICE_NAME WiFi.getHostname() // Device name is automatic from MAC-Address
#define DEVICE_NUMBER 0
#define SERIAL_SWAP
```

Here you define options in software:

- *if you use thinger.io or not,*
- *where you want to redirect your user terminal (Telnet, Serial, SoftSerial, on ESP32 also Serial2)*
- *if you are using inter-ESP communication and if it is a sender or receiver, the corresponding IP adresse and port,*
- *if you want to send reports to a computer, the corresponding IP adresse and ports*
- *If you want to force a hostname or use the default ESP-ABCDEF (ABCDEF being the 6 last hex values of the MAC address of the module.*
- *If you want to swap the serial port of an ESP8266 to use the sole UART on D7,D8*

```
//----- MEASUREMENT AND I/O OPTIONS-----
#define WEATHER_SOURCE_IS_OWM // _NONE , _OWM , _BME680 // (Source of weather Information, URL in Credentials.h)
#define A0_IS_NONE // _NONE , _INPUT, _DEMO
#define A0_MAX 40 // if A0 is used, define the value of the full range measure
#define A0_HALF 20 // if A0 is used, define the value of the half range measure (if different from A0_MAX /2 then with offset)
#define D0_IS_NONE // _NONE , _RELAY1
#define D5_IS_NONE // _NONE , _RELAY2
#define D6_IS_NONE // _NONE , _RELAY3
#define D7_IS_NONE // _NONE , _RELAY4; _SERIAL
#define D8_IS_NONE // _NONE , _RELAYS; _SERIAL
#define INA_IS_NONE // _NONE , _226 when measures come from an INA226 power sensor (Smart shunt scenario)
#define ADS_IS_NONE // _NONE , _1115 when measures come from an ADS1115 ADC converter (Drok type)
```

Here you define options for measurements:

Weather source _IS_OWM (Open Weather Maps API, connection details in credentials)

_IS_BME680 (Using a BME680 environment sensor over I2C)

A0_IS_NONE A0 not used

A0_IS_INPUT Used as input (the 2 next parameters define the conversion range and half range)

A0_IS_DEMO A0 is the light source for a solar charger demo (recommended to learn how to use all the features)

A0_MAX and A0_HALF (to convert the 0..1024 ADC in a measuring value; the half value is useful when a bias exists)

D0 ...D8 Define what you do with the IO. (Special case for D7,D8 which can be used for the redirected serial UART)

INA_IS_226 When you use an INA226 power measurement chip

ADS_IS_1115 When you use an ADS1115 precision 4 channel ADC

```
//----- WiFi Options -----
#define WIFI_REPEAT 500 //mS for retries and LED blinking during WiFi initialization
#define WIFI_MAXTRIES 30
#define NTP_DELAY 6000 //mS until NTP is settled (some NTP servers need time to react)
#define GRACE_PAUSE //Suspend Network processing for a grace pause, if the remote server takes too long to react, in order to keep being reactive on menus
```

Here you define options forWiFi initalisation:

WIFI_REPEAT the number of mS between attempts (and the half LED blink rate)

WIFI_MAXTRIES the number of attempts for connection to WiFi

NTP_DELAY the delay until the NTP time is finalized

GRACE_PAUSE when defined and the communication with thinger.io is wonky, blocking processing, introduce a pause to let the other functions become processed.

```
//-----Options to reduce WiFi power hence the current draw-----
#ifdef ARDUINO_ARCH_ESP8266
#define WIFI_POWER 7.5 // from 0.5 to 21.5 full power (more current draw)
#else
// ESP32
#define WIFI_POWER WIFI_POWER_11dBm
```



```
#endif
/*
Available ESP32 RF power parameters:
WIFI_POWER_19_5dBm // _19dBm, _18_5dBm, _17dBm, _15dBm, _13dBm, _11dBm, _8_5dBm, _7dBm, _5dBm, _2dBm, _MINUS1dBm
Available ESP8266 RF power parameters: any value in 0.5 steps from
0 (for lowest RF power output, supply current ~ 70mA) to 20.5 (for highest RF power output, supply current ~ 80mA)
*/
```

Here you may reduce the transmission power of the WiFi sender to reduce current draw.

```
//-----DO NOT EDIT until you know what you do -----
#define SERIAL_SPEED 9600
#define RELAY1 16 // D0 Relay or FET control 1
#define RELAY2 14 // D5 Relay or FET control 2
#define REDLED 15 // D8
#define BLULED 13 // D7 <-- Victron
#define GRNLED 12 // D6
```

CREDENTIALS.H

This tab is dedicated to your personal access codes.

```
#define WIFI_SSID "Secret" // Change it your own SSID
#define WIFI_PASS "Secret" // Change it your own Password

// ***Time zones***
#define NTP_SERVER "de.pool.ntp.org" // Change de. to your own country
#define TZ 1 // (utc+) TZ in hours
#define DST_MN 60
#define GMT_OFFSET_SEC 3600 * TZ
#define DAYLIGHT_OFFSET_SEC 60 * DST_MN

//Thinger
#define THINGER_USERNAME "Secret" // Change it your own UserName
#define THINGER_DEVICE_CREDENTIALS "Secret" // Change it your own Credential
#define THINGER_DEVICE DEVICE_NAME
```

Here you enter your connection information to thingsr.io

You define the device credentials in thingsr.io for each device and must match.

The last line should remain unchanged unless you want a (confusing) device name different from the host name.

```
//Location for weather
#define OPEN_WEATHER_MAP_API_KEY "Secret" // Change it your own ID
#define OPEN_WEATHER_MAP_LOCATION_ID "TownCode" // Change it your own TownCode
#define OPEN_WEATHER_MAP_LANGUAGE "en"
#define OPEN_WEATHER_MAP_UNITS "metric" //Americans: learn Metric ;-)
```

Here you enter your connection information to <https://openweathermap.org/>

You can create a few API keys free of charge https://home.openweathermap.org/api_keys

You get your location ID here: <https://openweathermap.org/city/2928810> the code is the numeric part of the URL.

A_LIBS_VARS

The two above mentioned tabs are what you need to configure. To start e.g. with a demo you don't need to change any tab further.

With a_Libs_Vars begins the code itself

In this tab the requested libraries are invoqued and the variables used across tabs defined.

A specificity is some variable definitions as structures e.g.

```
struct payload {
  /***Operating Values from Victron/SmartShunt***
  float BatV; // V Battery voltage, V
  float BatI; // I Battery current, A
  float BatW; // W BatV*BatI

  float PanV; // VPV Panel voltage, V
  float PanI; // PanW/PanV
  float PanW; // PPV Panel power, W

  float LodI; // IL Load Current A
  float LodW; // BatI*BatV
  float IOhm; // dV / dI

  int ChSt; // CS Charge state (not POC), 0 to 9
  int Err; // ERR Error code, 0 to 119
  boolean LodOn; // LOAD ON Load output state, ON/OFF
} payload;
```

A structure can be easily transmitted to another ESP over WiFi enabling to split the functionality between two ESP modules. E.g. the first ESP processes the data acquisition outdoors and sends the structure to another indoor ESP module, that displays the data and processes reports.

B1_KARAJAN_FUNCTIONS

This tab contains all functions used by the basic features of Karajan to initialize WiFi, time and provide the timing variables.

B2_USER_FUNCTIONS

Place here you own functions

C_SETUP

This tab contains the initialization procedure to start the ESP with WiFi, Time and initialize the periphery.

The setup procedure is verbose on USB-Serial. You should note some important values during the first run:

```
16:53:29.458 -> Compiled from: D:\Activities\3_Maker\ESP_VictronDrok_on_Steroids_V05.2024\c_Setup.ino
16:53:29.526 -> on Apr 26 2024 at 10:13:36
16:53:29.559 -> Victron/INA226 Logger at work: esp32-ABCDEF Serial @ 9600 Baud
16:53:29.628 -> Reconnecting to GW-FM-7390
16:53:35.096 -> Connection OK!
16:53:35.096 -> Done: RRSI= -57dBdB, IP= 192.168.188.037
16:53:43.095 -> Got Epoch, Now: 16:53, 29 April 2024
16:53:43.132 -> Start OTA, Menu ready on Telnet
16:53:44.120 -> Serial start on GPIO16,17
```

The **host name**, which is also the device name at tinker.io

The **IP address** (your router should assign permanent or very long term IP leases to get a remote access to your module).

Please also verify that the time info is correct, it is important for reliable statistics.

If a screen is available the host name and the IP Address are displayed:



If the WiFi credentials are already known, the procedure is short: the blue onboard LED will only flash briefly.

If the internal WiFi credentials do not match, the boot procedure will make a second attempt with the credentials defined in [Credentials.h](#)

During that attempt, the blue onboard LED will flash at a one second pace.

If that attempt is not successful, the boot procedure will invoke Smart Config / ESP Touch.

https://www.espressif.com/en/support/download/apps?keys=&field_technology_tid%5B%5D=20

During SmartConfig, the blue onboard LED will flash at a three second pace.

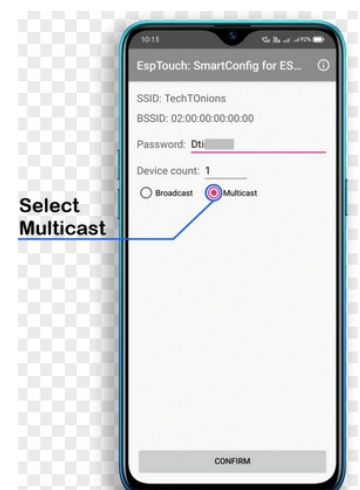
Smart Config is a free Android or iOS app in which you can enter the WiFi credentials, which are then transmitted to your ESP device.

You just select the WiFi SSID, enter your password and confirm.

```
Compiled from: D:\Activities\3_Maker\ESP_Karajan_Demo05_2024\c_Setup.ino
on Apr 30 2024 at 15:56:46
Karajan at work: Karajan Serial @ 9600 Baud
Reconnecting to GW-FM-7390
Fail, try Credentials
```

```
.....Fail, try SmartConfig
```

```
.....Connection OK!
Done: RRSI= -58dBdB, IP= 192.168.188.060
Got Epoch, 16:00 30Apr
Start OTA, Menu ready on Telnet
```



D_MENU

This tab contains the menu processing code defined in the chapter [Menu](#)

The code is largely self-explaining.

It contains also the code to process the buttons and eventually rotary knob of a TTGO module to change the display pages and the corresponding setpoints

E1_INSTANT_DATA

This part is processed with the highest priority. It should only contain asynchronous communication tasks that are triggered from an external device

Be very careful to place here only code that is periodically triggered from another device and reliably finishes without delay.

E2_FAST_DATA

This part is processed every 128mS, it contains a call to [thinger.io](#) to update the dashboard, demonstration that you can have very fast dashboards, ways faster than over MQTT.

It contains also the analog input processing of A0, with an example of oversampling the input reducing noise.

Both features can be moved to the next tab, if you don't need the speed.

E2_SLOW_DATA

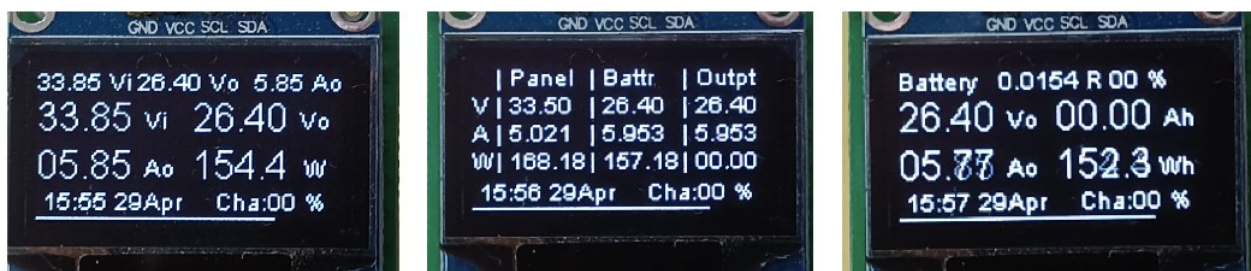
This part is processed every second, it contains the simulation of a solar charger for didactic purposes and to provide some life to the display and reports. Of course, once you have got some training, you are encouraged to replace that code with your own functionality.

F_STATS

This part is processed every second, immediately after the slow data. It contains an example of the code to compute time-related statistics. That code is left for didactic purposes only, you are encouraged to replace it with your own functionality.

G_DISPLAY

This part is processed every second, It contains an example of the code to display operating values on OLED screens driven by an I2C interface:



All displays include a header overview and a footer time line.

On the color screen of TTGO LilyGo devices, the information is ways more pleasant:



You can switch between the display screens using the 2 hardware buttons of the TTGO module, or over the Telnet Menu using your computer by typing 1 to 6 (as it can be seen on the monitor behind the TTGO modules.)

If the TTGO module is equipped with a rotary knob, you may change the screens upon turning the knob as well.

H_SERIAL

This part is processed every second, It contains an example of the code to generate the reports shown on the chapter [Menu](#)

I_WIRELESS

This part is processed every second, It contains an example of the code to

- communicate with Dashboards
- communicate between ESP devices*
- send periodical reports to a computer.

The communication between ESP devices is simple and damn efficient: the structure “payload” is converted byte-to byte into a string of chars (the technique is called data punning). This string is sent over UDP to the other ESP.

When the other ESP receives a string over UDP which has the same byte length it is converted back to the same payload structure.

That way, the receiving ESP gets the same payload structure and can process further the data as if it were it’s own acquisition.

Since the data transmitted has no privacy value (it’s just a raw flow of measure bytes) I considered that an encryption is not necessary.

S_SCHEDULER

The scheduler is the main loop() of the Arduino sketch.

It calls the other tabs in a timely manner. The one second paced tabs are called in a distributed way over the second so that not everything is processed at the same time.

The schedule provides also the time flags described at chapter

T1_FRONTPAGE_THINGER.H

T2_STATISTICS_THINGER.H

These two tabs do not contain ESP code and are not compiled. They contain the JSON structures describing the dashboards examples at [thinger.io](#). They are given for reference and can be used to start with ready-made dashboards.

THINGER.IO DASHBOARDS

Everyone can register free of charge* up to two devices to plot information and design dashboards in a very versatile way.

** It will remain so forever: of course Thingier.io hopes that you will want more and get e.g. a maker plan subscription with up to 25 devices and dashboards or even an own Cloud server with unlimited resources, but you can do quite a lot with the free accesses...*

Thingier.io delivers:

- Fast dashboards that displays data down to a 1/8 second periodicity (MQTT mostly limits you to 1 min).
- Real-time trends and historical trends
- Statistic dashboards that display information gathered by thingier.io over long time.
- Data buckets that gathers periodical information over up to 365 days, from which you can download CSV data to e.g Excel.
- Properties to store and retrieve values from your devices
- Rules to forward warnings over e.g. eMail.
- Interact with the device to e.g. drive a relay or initiate calibration
- and much, much more...

CREATE AN ACCOUNT

First create a free account at thingier.io: <https://console.thingier.io/login>

SETUP A DEVICE

Set up device as follows : <https://console.thingier.io/#!/console/devices/add>

Use the device name reported at chapter [c_Setup](#) and the credentials defined at chapter [Credentials.h](#)

SETUP RESOURCES AND PROPERTIES

Within a device you have

- “Resources” instant values, written by the device or polled by a dashboard
- “Properties” persistent, written /read back by the device or polled by a dashboard

containing the values of your devices.

These are defined by the program, you don’t have to create them manually.

Properties have the advantage over resources that they store values permanently and can be retrieved by the program as would an external memory or an EEPROM do.

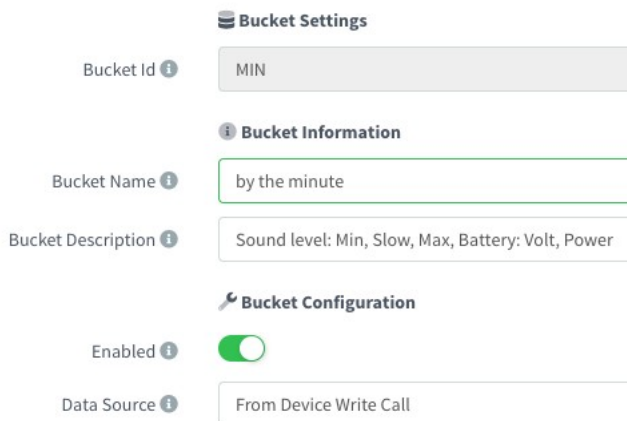
BUCKETS

Additionally, across devices, you have the so-called buckets, used to store long-term data for reporting and to draw timeseries charts. You have to create the buckets manually for your project.

Set up the following “data buckets”

- MIN. used for Minute recording
- EVENT used for asynchronous events like Battery low, Battery full,
- DAY used for Daily report – Ah summary, Battery voltage summary
- HOUR used for Hourly Data battery values, Weather

The bucket ID must match the 4 given names, the bucket name and bucket description can be freely chosen..



Bucket Settings

Bucket Id ⓘ MIN

Bucket Information

Bucket Name ⓘ by the minute

Bucket Description ⓘ Sound level: Min, Slow, Max, Battery: Volt, Power

Bucket Configuration


Enabled ⓘ ☒

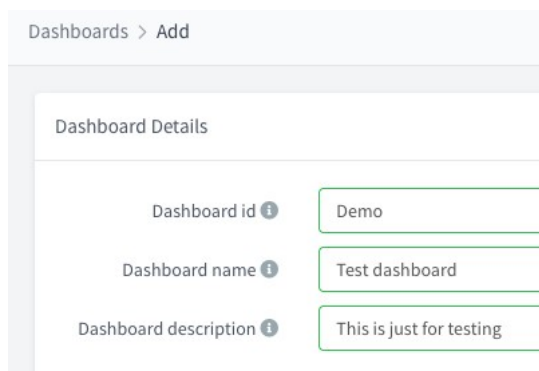
Data Source ⓘ From Device Write Call

DASHBOARDS

With a free account, you may create up to 4 dashboards.

In a dashboard you place “widgets”. You can add / change remove widgets any time, right from the dashboard by entering the configure mode at the top right of the dashboard.

The widgets will then show  to become duplicated, configured, deleted



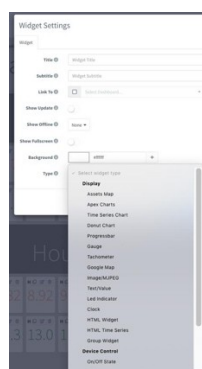
Dashboards > Add

Dashboard Details

Dashboard id ⓘ Demo

Dashboard name ⓘ Test dashboard

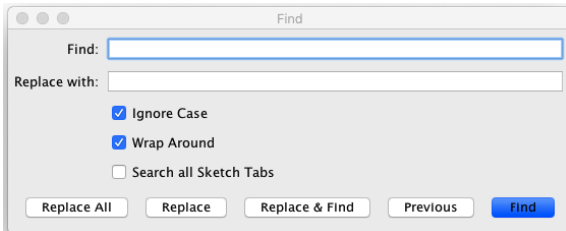
Dashboard description ⓘ This is just for testing



To start with ease, Karajan provides two preconfigured dashboards cf. chapter t1_Frontpage_Thinger.h t2_Statistics_Thinger.h.

You must however first adjust the device name and your user name before using it.

Using the editor of Arduino IDE :



- replace every occurrence of `<"id": "Karajan">`, with your device name.
(You may skip that, if you configured Config.h with the device name "Karajan")
- replace every occurrence of `"user": "rin67631"` with your own user name at thinger.io.

Select all text of that file and place it into your clipboard.

In <https://console.thinger.io/console/dashboards> open your newly created dashboard. click on Settings, then on Developer.

replace the text in the frame JSON with the content of your clipboard. Save.

Dashboard Settings

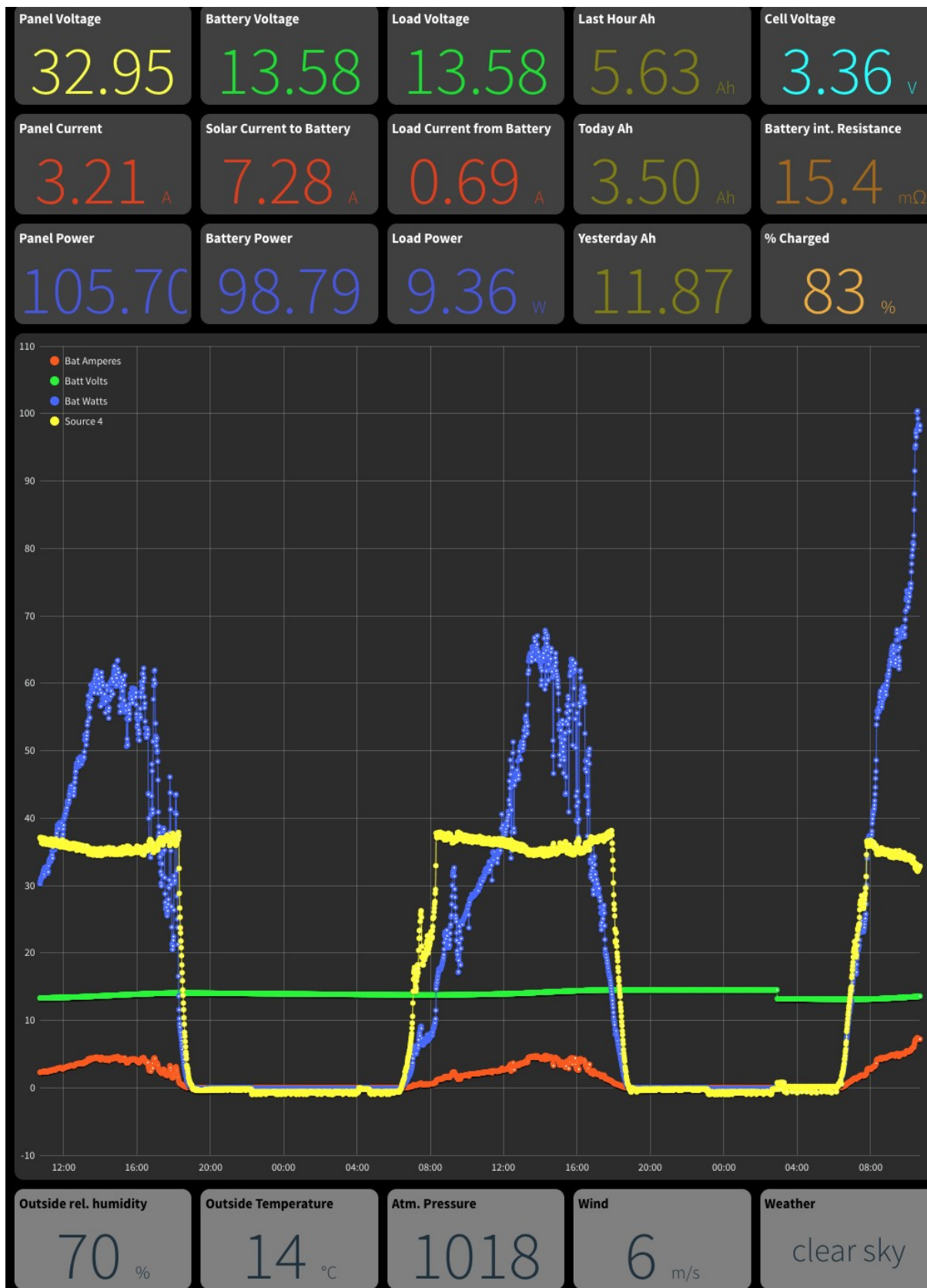
Layout Share Developer Placeholders Functions Controls

Dashboard Configuration

{ } JSON

```
{
  "description": "Statistics",
  "name": "Statistics",
  "placeholders": {
    "sources": []
  },
  "properties": {
    "background_image": "#BBBBBB",
    "columns": 12
  },
  "tabs": [
    {
      "icon": "fas fa-tachometer-alt",
```

Admire your new dashboard:



You can do the same with the next dashboard preset: `t2_Statistics_Thingier.h`:

Hourly statistics					Other statistics	
	Current Day		Yesterday			To/from Battery
From00:00to00:59	-2.10 Ah	28.16 v	-2.08 Ah	28.16 v	Last Hour	4.79 Ah 28.03 v
From01:00to01:59	-2.09 Ah	28.16 v	-2.09 Ah	28.16 v	Today	33.22 Ah 28.09 v
From02:00to02:59	-2.10 Ah	28.16 v	-2.09 Ah	28.16 v	Yesterday	45.40 Ah 28.09 v
From03:00to03:59	-2.10 Ah	28.16 v	-2.08 Ah	28.16 v	D-2	-2.08 Ah 28.16 v
From04:00to04:59	-2.10 Ah	28.16 v	-2.09 Ah	28.16 v	D-3	40.68 Ah 28.09 v
From05:00to05:59	-2.10 Ah	28.16 v	-1.98 Ah	28.14 v	D-4	42.18 Ah 28.00 v
From06:00to06:59	-1.92 Ah	28.13 v	1.43 Ah	28.08 v	D-5	-2.08 Ah 27.92 v
From07:00to07:59	2.07 Ah	28.06 v	3.67 Ah	28.06 v	D-6	45.33 Ah 26.85 v
From08:00to08:59	4.10 Ah	28.06 v	4.30 Ah	28.06 v	D-7	46.52 Ah 26.42 v
From09:00to09:59	4.26 Ah	28.06 v	4.38 Ah	28.05 v		
From10:00to10:59	4.40 Ah	28.05 v	4.38 Ah	28.06 v		
From11:00to11:59	4.37 Ah	28.06 v	4.61 Ah	28.05 v		
From12:00to12:59	4.43 Ah	28.05 v	4.63 Ah	28.05 v		
From13:00to13:59	4.60 Ah	28.05 v	4.74 Ah	28.05 v		
From14:00to14:59	4.82 Ah	28.05 v	4.71 Ah	28.06 v		
From15:00to15:59	4.74 Ah	28.05 v	4.45 Ah	28.05 v		
From16:00to16:59	5.15 Ah	28.04 v	4.87 Ah	28.04 v		
From17:00to17:59	4.79 Ah	28.03 v	4.62 Ah	28.04 v		
From18:00to18:59	4.45 Ah	28.04 v	4.45 Ah	28.04 v		
From19:00to19:59	4.85 Ah	28.05 v	4.85 Ah	28.05 v		
From20:00to20:59	3.49 Ah	28.10 v	3.49 Ah	28.10 v		
From21:00to21:59	-1.56 Ah	28.16 v	-1.56 Ah	28.16 v		
From22:00to22:59	-2.10 Ah	28.16 v	-2.10 Ah	28.16 v		
From23:00to23:59	-2.10 Ah	28.16 v	-2.10 Ah	28.16 v		

Of course these are only starters matched to my demo, unleash your creativity on your own application.

CONCLUSION

The beauty of ESP devices, with thingier.io and ESP Karajan is that you can get all that stuff out of a single 3\$ device, without any other hardware required: no computer running permanently at home, no MQTT server, no Grafana, no Influx database to configure, no hassle with matching everything together.

WHICH DEVICES ARE SUPPORTED?

Following devices are fully supported:

- ESP8266: Lolin Wemos D1, D1Pro, clones, Generic ESP8266 “F” Modules, Adafruit feather Huzzah.... For D1 mini: OLED 64x48 and OLED 128x64 OLED display with I2C interface.
- ESP32: Lolin Wemos 32, Lolin Wemos 32 OLED128x64, TTGO/LILYGO with TFT display, HELTEC LoRa with 128x64 OLED display.

Others may work out of the box or need some minor adaptations.

BEGINNER'S GUIDE

INSTALLING THE IDE

Install Arduino IDE 2.0, if you don't already have it: <https://www.arduino.cc/en/software>

CONFIGURE THE IDE

In the menu Preferences, enter the address of the additional boards' files:

http://arduino.esp8266.com/stable/package_esp8266com_index.json , https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series/releases/download/1.0.0/package_heltec_esp32_index.json ,
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Take note of your sketchbook location.

INSTALL THE BOARDS AND LIBRARIES ONTO THE ARDUINO IDE

INSTALL THE BOARDS AND LIBRARIES ON THE ARDUINO IDE

BOARDS:

- esp8266 by ESP8266 Community
- esp32 by Espressif Systems (if you plan to use an ESP32 device)
- Heltec ESP32 Series Dev-boards by Heltec (if you plan to use a Heltec / LoRa device)

GLOBAL LIBRARIES

- ESP8266 and ESP32 OLED driver for SSD1306 displays by ThingPulse, Fabrice Weinberg
- MobaTools by MicroBahner from which I took the Motobuttons.h lib
- Arduinojson by Benoit Blanchon <blog.benoitblanchon.fr>
- ThingyCore32 and Thingy.io by Alvaro Luis Bustamante
- TelnetStream by Juraj Andrassy
- InterpolationLib by Luis Llamas
- INA226 by Rob Tillaart (if you plan to use the vendor-agnostic shunt version)
- TFT_eSPI by Bodmer (if you plan to use the TTGO remote color display)

N.B. this library must be configured specifically to run for the TTGO !

Once installed, you must replace the file

[your sketchbook folder] \libraries\TFT_eSPI\User_Setup.h" with the file User_Setup.h from
https://github.com/rin67630/Victron_VE_on_Steroids/blob/main/Config_Files/User_Setup.h

PROJECT'S LIBRARIES

We will place there the Hardware Abstraction Layer files containing the definitions for specific Victron modules and Shunt characteristics.

Under your sketchbook location, subfolder libraries create a subfolder named /Karajan_HAL_files/.

Copy the files from https://github.com/rin67630/ESP_Karajan/tree/main/Config_Files into that folder.

GET THE ARDUINO IDE FILES FROM GITHUB

https://github.com/rin67630/ESP_Karajan/tree/main/Software

From there press on the green Code button and chose Download Zip.

Store the file *** to the folder downloads. Unzip it. Copy the folder Software to your sketchbook location.

Rename that folder with the name of the file beginning with “ESP_Victron_on_Steroids_V”

Start your Arduino IDE, open the corresponding sketch.

The IDE of Karajan will display many folders. Cf Chapter: [The Arduino IDE: Tabs](#)

CONFIGURE THE SOFTWARE

Cf Chapter: [Configuration](#)

N.B. Enter the parameters exactly as shown in CAPITALS and with UNDERLINES where indicated. Else the parameter will be ignored and the compilation may fail or give wrong results.

COMPILE YOUR SKETCH

Once all data has been entered, you are almost ready to compile your sketch.

If you have chosen DASHBRD_IS_THINGER (the regular case) , you must before have defined the thinger.io device and buckets configuration (next chapter).

Before compiling, make sure your compiler is set to the right board type:

- For Witty, WEMOS D1 min, WEMOS8266 chose the board Wemos D1 (Clone).
- For WEMOS 32 chose ESP32 WROOM-DA Module
- For TTGO or Lilygo, chose ESP32 TTGO-T1
- For HELTEC-LoRa chose Heltec Wi-Fi oRa32 (not V2!)

For the first time, your board must be connected through its serial port, and you should activate the serial monitor (set to the Baud-rate defined in Config.h).

If you get compiling errors, make sure that all libraries are present and your parameters are matching your hardware.