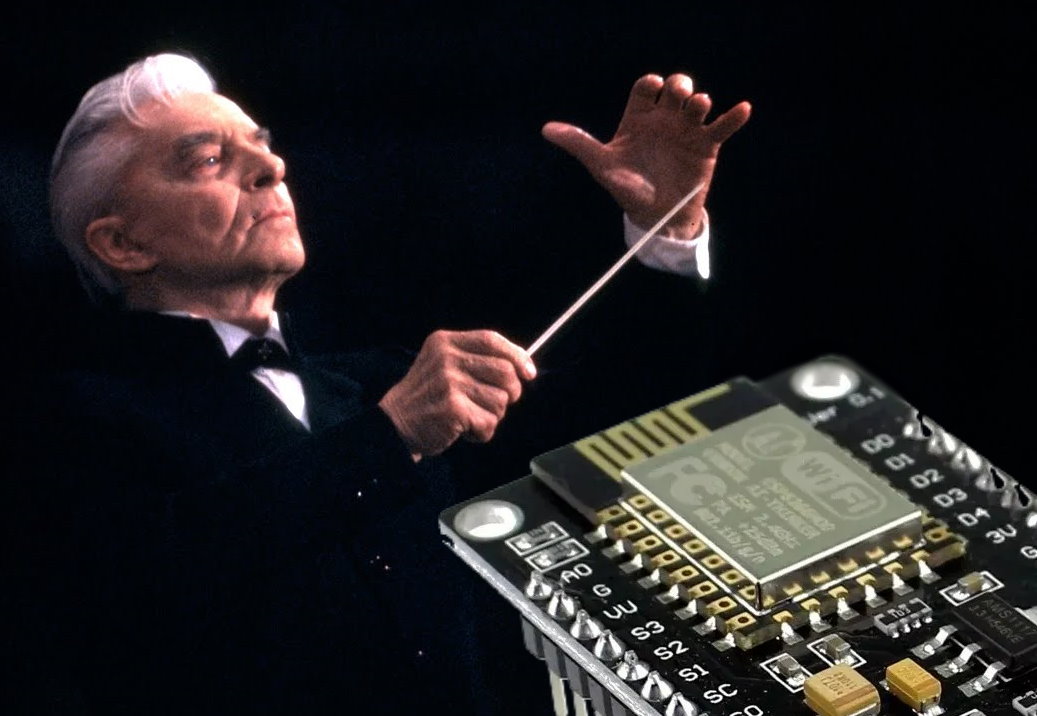
ESP Karajan

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](#_Toc165388681)

[What is ESP-Karajan? 3](#_Toc165388682)

[Scheduler 3](#_Toc165388683)

[Communication 3](#_Toc165388684)

[Menu 4](#_Toc165388685)

[Dashboards 5](#_Toc165388686)

[Non-Volatile Memory 5](#_Toc165388687)

[Inter-ESP Communication (option) 5](#_Toc165388688)

[Computer Communication (option) 5](#_Toc165388689)

[The Arduino IDE: Tabs 6](#_Toc165388690)

[What is ESP-Karajan not? 6](#_Toc165388691)

[Configuration 7](#_Toc165388692)

[Config.h 7](#_Toc165388693)

[Credentials.h 8](#_Toc165388694)

[a\_Libs\_Vars 8](#_Toc165388695)

[b1\_Karajan\_Functions 9](#_Toc165388696)

[b2\_User\_Functions 9](#_Toc165388697)

[c\_Setup 9](#_Toc165388698)

[D\_Menu 10](#_Toc165388699)

[e1\_Instant\_Data 10](#_Toc165388700)

[e2\_Fast\_Data 10](#_Toc165388701)

[e2\_Slow\_Data 10](#_Toc165388702)

[f\_Stats 10](#_Toc165388703)

[G\_Display 10](#_Toc165388704)

[H\_Serial 12](#_Toc165388705)

[I\_Wireless 12](#_Toc165388706)

[S\_Scheduler 12](#_Toc165388707)

[t1\_Frontpage\_Thinger.h t2\_Statistics\_Thinger.h 12](#_Toc165388708)

[Thinger.io dashboards 13](#_Toc165388709)

[Create an account 13](#_Toc165388710)

[Setup a device 13](#_Toc165388711)

[Setup resources and properties 13](#_Toc165388712)

[Buckets 13](#_Toc165388713)

[Dashboards 14](#_Toc165388714)

[Preconfigured dashboards 15](#_Toc165388715)

[Conclusion 17](#_Toc165388716)

[Which devices are supported? 17](#_Toc165388717)

[Beginner’s Guide 18](#_Toc165388718)

[Installing the IDE 18](#_Toc165388719)

[Configure the IDE 18](#_Toc165388720)

[Install the boards and libraries on the Arduino IDE 18](#_Toc165388721)

[Boards: 18](#_Toc165388722)

[Global Libraries 18](#_Toc165388723)

[Project’s libraries 18](#_Toc165388724)

[Get the Arduino IDE Files from GitHub 19](#_Toc165388725)

[Configure the software 19](#_Toc165388726)

[Compile your sketch 19](#_Toc165388727)

# 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 integation values (here Ah)
* J. (upper case) Display the job execution time monitoring evey 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 EEPPROM 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

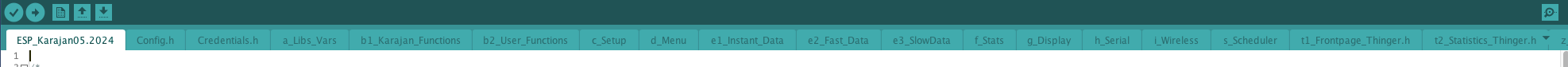
### Inter-ESP Communication (option)

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 , \_RELAY5; \_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 menues

*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 thinger.io   
You define the device credentials in thinger.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/*](https://openweathermap.org/) *You can create a few API keys free of charge* [*https://home.openweathermap.org/api\_keys*](https://home.openweathermap.org/api_keys) *You get your location ID here:* [*https://openweathermap.org/city/2928810*](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

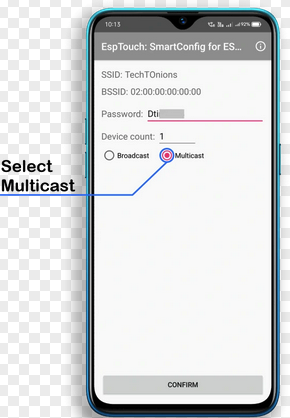
16:53:43.132 -> Start OTA, Menu ready on Telnet

16:53:44.120 -> Serial start on GPIO16,17

The host name, which is also the device name at thinger.io  
The IP address (your router should assign permanent of very long term IP leases to get a remote access to your module.  
Pease 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 invoque 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 ony contain asynchonus 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.

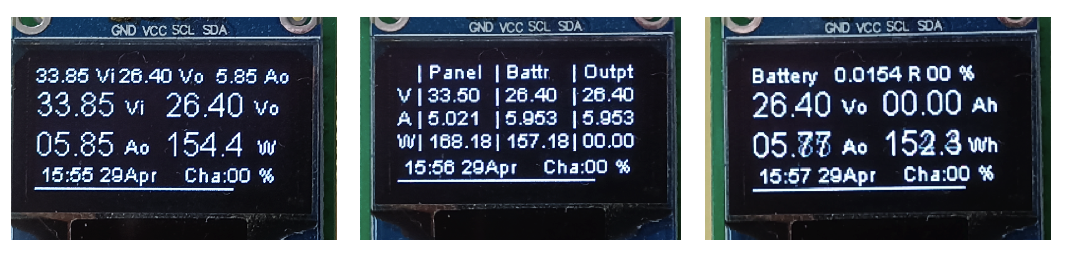
### e2\_Slow\_Data

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 Scheduler

### 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 Thinger.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...*

Thinger.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 thinger.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 thinger.io: https://console.thinger.io/login

### Setup a device

Set up device as follows : https://console.thinger.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.

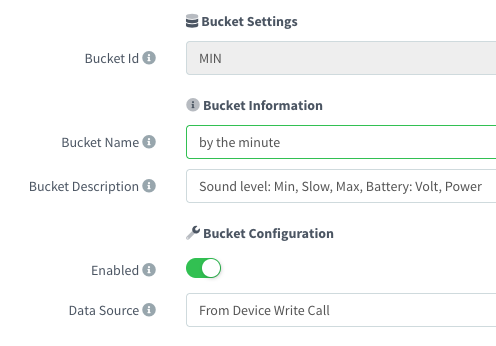
### Buckets

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..

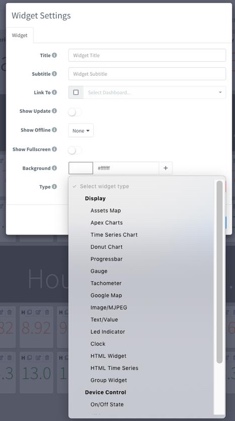


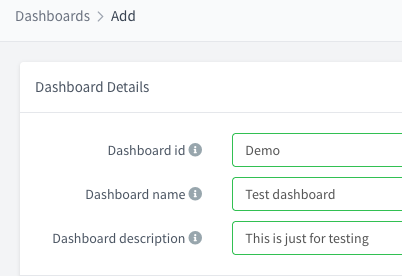
### Dashboards

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

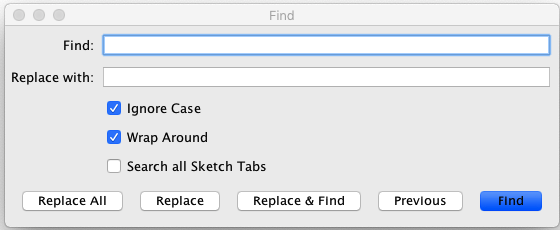
In a dashboard you place “widgets”. You can add / change remove widgets any time, right from the dashboard by entering the configure mode at the top right of the dashboard.

The widgets will then show page25image40552480to become duplicated, configured, deleted





### Preconfigured dashboards

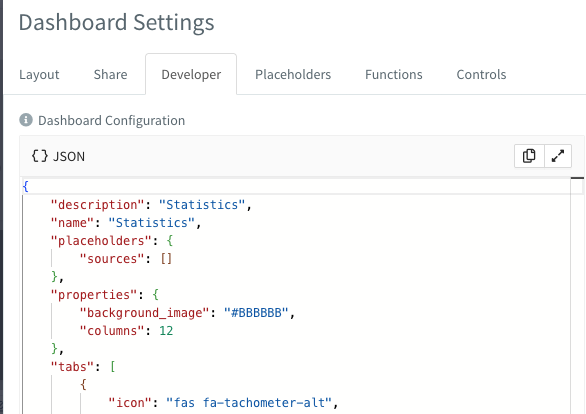
To start with ease, Karajan provides two preconfigured dashboards cf. chapter t1\_Frontpage\_Thinger.h  
t2\_Statistics\_Thinger.h.  
You must however first adjust the device name and your user name before using it.  
Using the editor of Arduino IDE :   
  


* 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.

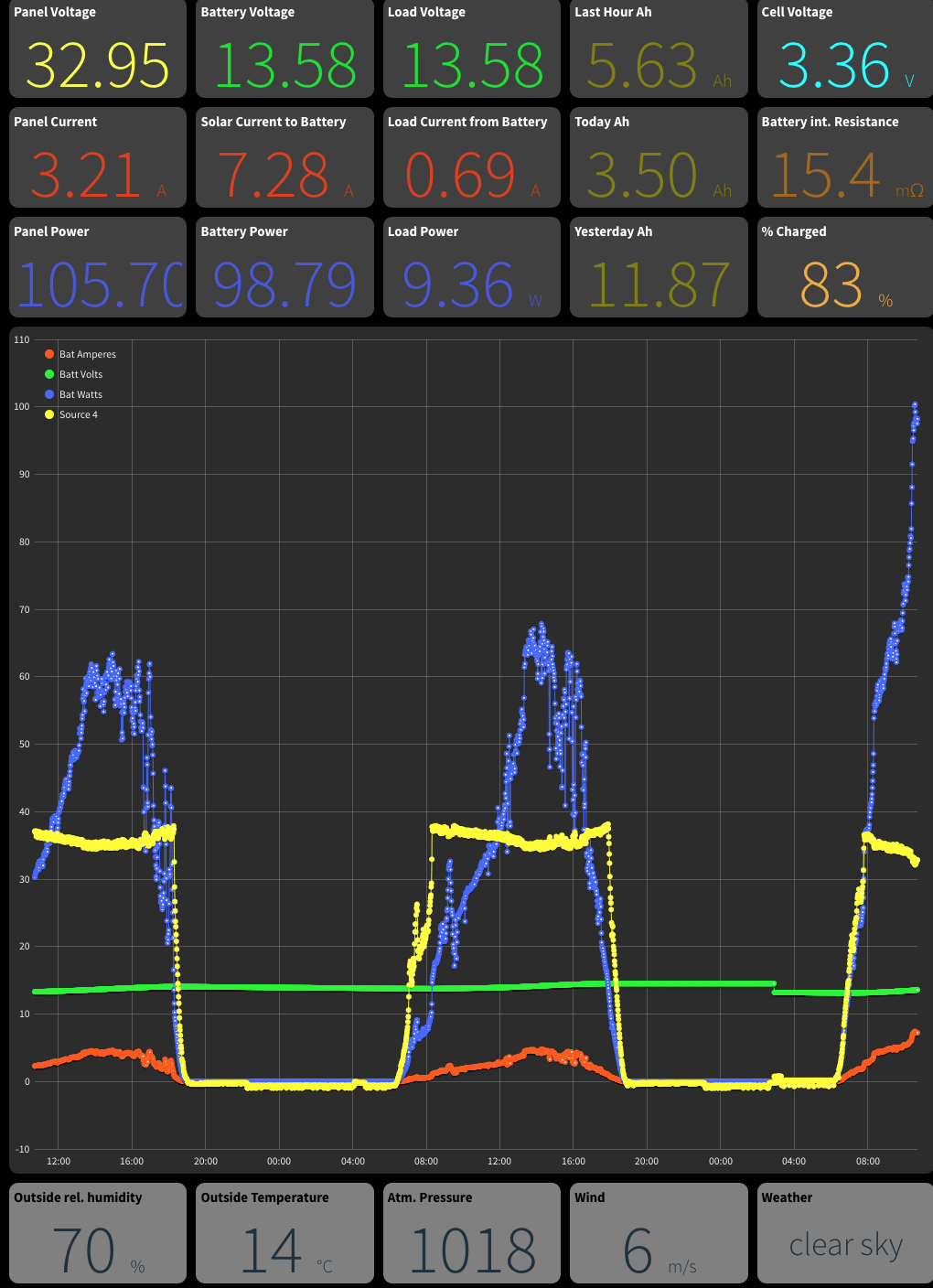
Select all text of that file and place it into your clipboard.

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

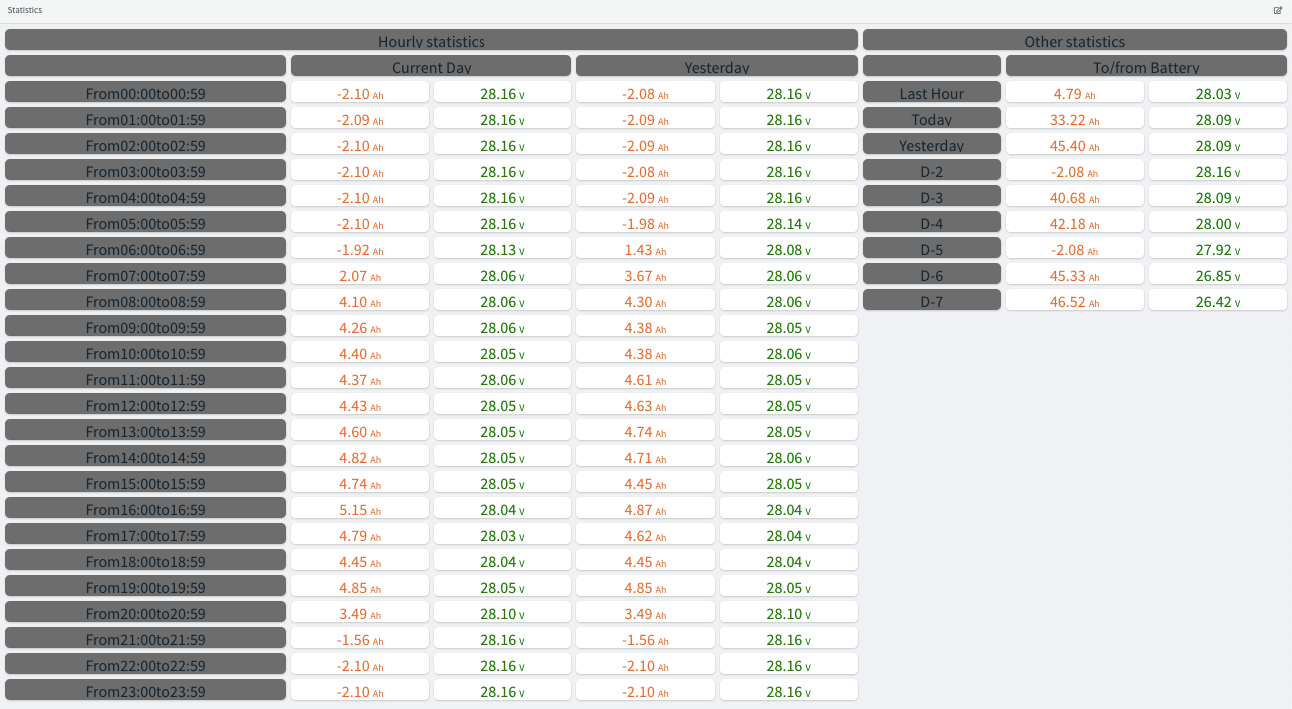
replace the text in the frame JSON with the content of your clipboard. Save.

page26image56686592page26image42770496

Admire your new dashboard:



You can do the same with the next dashboard preset: t2\_Statistics\_Thinger.h:



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

## Conclusion

The beauty of ESP devices, with thinger.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>
* ThingerCore32 and Thinger.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 WiFiL 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.