LoRa Beacon
HW/SW Installation and User manual
Vr. 5.1

Issue 1.0
Issue date: 08/11/2023

Author: Michele Fucito, *I8FUC*
Translation&Revision: Alfredo Vania *IZ7BOJ*

CONTENT INDEX

FIGURES INDEX

ACRONYMS

ACRONYM	Explanation
AP	Access Point
APRS	Automatic Position Reporting System
APRS-IS	APRS Internet Service
BT	Bluetooth
BW	Band Width
CPU	Central Processing Unit
CR	Coding Rate in case of Lora Protocol or Carriage Return
DHCP	Dynamic Host Configuration Protocol
EEPROM	Electrically Erasable Programmable Read-Only Memory
FRAM	Ferroelectric RAM
GPS	Glopal Positioning System
GUI	Graphic User Interface
HW	Hardware
IoT	Internet of Things
IP	Internet Protocol
KISS	Keep It simply Stupid protocol
LAN	Local Area Network
LF	Line Feed
LoRa	“Long Range” Radio Communication Technique
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport Protocol
OLED	Organic Led (display technology)
OTA	Over The Air update
PCB	Printed Circuit Board
PPM	Parts Per Million
PPS	Pulse Per Second
RAM	Random Access Memory
RTC	Real Time Clock
RX	Receive
SDR	Software Defined Radio
SF	Spreading Factor
SMD	Surface Mounted Devices
SNR	Signal to Noise Ratio
SPI	Serial Peripheral Interface
SW	Software
TCXO	Temperature Compensated Crystal Oscillator
TFT	Thin-film transistor (display technology)
TX	Transmit
uC	MicroController

1 Introduction

This document describes the HW assembly and SW installation methods of the devices based on the LoRa_Beacon project developed in the context of the SARIMESH experimentation (ref. https://www.sarimesh.net).

The goal of this project is to generate an HW/SW platform that can be used for experimentation activities **relating to LoRa technology applied to radio amateur** and therefore not necessarily using the guidelines and implementations mandatory in the IoT (Internet of Things) applications.

It is worth remembering that LoRa technology was born mainly to allow the creation of "sensor" type devices or for M2M (Machine-To-Machine communication). This type of applications is characterized by a limited communication need in terms of quantity of transmitted data and by a maximum independence from external power sources, trying to use batteries to power the devices with an autonomy target of the order of years.

As a result of these functional requirements, the LoRa radio protocol has aimed to reduce the transmission power of the devices according to the available **"radio link budget"**; the key to achieve this objective is the use of a spread spectrum transmission method which allows to obtain a significant “**process gain**”.

Therefore, in the context of the IoT, the reduction of energy consumption assumes enormous importance, considering their use as remote sensors not powered by external energy sources.

In amateur radio use, this aspect obviously takes a back seat, while the aspects of maximizing performance in terms of "**available radio link budget**" take on particular importance.

It was therefore decided in our experimentation to neglect, at least initially, the aspects of minimization of energy consumption, which are inessential for the purposes of our uses.

The target user has therefore been thought to be the average radio amateur eager to experiment with this new HW/SW technology with little expense and without particular skills.

The chosen solution is based on the reuse of **HW and SW "functional blocks"** already available on the market and on the internet.

2 Needed Hardware

The HW platform can be:

- A Sarimesh Custom HW device
- A classic "Chinese cards", currently widely used for LoRa applications and which still almost always mount first generation LoRa chips (es: TTGO T-Beam)

The Sarimesh HW is based on the concept of “mother and daughter boards”, which are small printed circuits that can be purchased on the classic e-commerce platforms at very convenient prices.

The daughter boards have been designed with different "footprints" which cover the most popular lora modules available on the market.

The first generation (year 2020) of LoRa Sarimesh HW devices was based on "Pin in Hole" technology as much as possible, limiting the use of SMD components only to exceptional cases. Also this choice was motivated to make the assembly of the circuits as easy as possible, without any particular manual and assembly skill.

With the second generation (year 2023) of Sarimesh HW devices the SMD use has been extended to cover most of the standard components, in order to gain valuable PCB areas for adding new modules on the PCB, leaving the "Pin in Hole" usage only for modules or mechanical components.

Unfortunately, the support of different HW platforms needs an economic commitment to obtain samples and a lot of time to carry out the non-regression tests.

With version 4.x we we decided to delegate the complete test of the SW on the TTGO boards due to the evidence that there were many different versions of this HW that are often quite different from each other.
With the present vr. 5.x SW version we implemented several additional features in order to allow easy installation of this SW also on the most diffused variants of the TTGO T-Beam platform; a special feature allows to customize via GUI the most frequent parameters to adapt to different sub-variants.

We tested our SW on TTGO T-Beam platform as best as possible on the samples we have available in our shacks, leaving the test on each specific device to the interested people, who will eventually be able to share their experiences via the GITHUB system.

The user can also build his own Hardware. In this case, it's possible to modify by GUI most of the parameters related to the HW architecture, as the I/O pins used by the ESP32 for interfacing to various chips installed on the board.

A 3D-printable enclosure for both tracker and iGate in "Sarimesh HW version", is also available on GitHub.

If a TTGO T-Beam board is used, a lot of cases or 3D printing files are already available on the internet.

At the end of this guide, the reader can also find a list of useful links for purchasing the required HW modules online at good prices.

3 Needed Software

The source code is available on Github, at this link:
https://github.com/IZ7BOJ/ESP32_LoRa_APRS_SARIMESH

From the SW point of view we tried to use a similar approach, using **"SW functional blocks"** already available in the open source environment, limiting new developments to the bare essentials.

A particular note in this regard is the programming style used: being the target a typical radio amateur, we tried to avoid "courtly" programming style and to write code easily understandable even by people who are not professional programmers.

The goal was and remains to encourage even the inexperienced radio amaterur to experiment by his hand. The use of particularly advanced programming techniques is the exact opposite of what is needed to lower the entry threshold to this type of applications.

The SW aspect is obviously closely linked to the HW choice of a specific type of processor used: the choice fell on the use of the **ESP32 processor** which in the field of microcontrollers represents an optimal solution as it provides an extremely low cost platform, very powerful and capable of operating in "multiprocessing" mode in "real time", supported by a very light and efficient operating system (FreeRTOS) and supported by **PlatformIO,** which became one of the most used platform for this type of application and, moreover, extremely more advanced than Arduino, which was used during the early stages of development.

A very important aspect taken into account during the development, is the device management**:** also in this case, we have tried to avoid potential users having to necessarily go through the complete installation of the SW development environment (process typically required by projects based on microcontrollers) by providing a simple, easily customizable graphical user interface (GUI) which allows to set many of the available functions and enjoy the experimentation.

In this document, some notes for the assembly of prototypse in the currently available Sarimesh HW PCB are presented, as well as the description of the SW image loading and its configuration.

At the end of the document, there is also a HW/SW "debugging" chapter, very useful if the user wants to join the development group, or if there are problems during the construction phase.

4 Compatibility with other existing LoRa APRS Standard

The Sarimesh SW supports both AX.25 encapsulation and the Austrian LoRa APRS implementations, which is a world-wide de-facto standard at present. We will call this mode "OE_Style" from now on; the SW can always decode APRS traffic in both AX.25 or OE_Style encapsulation; the outgoing LoRa traffic can be set by GUI in one way or another; so an OE-Style to AX.25 or viceversa adaptation is performed just in case.

The AX.25 encapsulation appears to be still an important feature in order to support easy integration of these HWs with standard APRS TNC devices using KISS interface; this allows to extend these existing TNC with the new LoRa transport and implement mixed LoRa/standard APRS nodes.

This feature allows you to use in the same LoRa APRS network both AX.25 and OE_Style devices.

5 Features introduced up to Version 5.x

With the Sarimesh SW 4.x and further 5.x version, new interesting features have been introduced; following is a resume (in time order...):

1) Use of standard **APRS compression for the geo-coordinate**s of transmitted packets, which can be enabled or not from the GUI. During reception, the SW can always decode both compressed and uncompressed location.

2) "**Agile Beaconing**": it's an algorythm for the optimization of beacon transmitting rate considering the trajectory of the moving tracker. The implemented algorithm can be set at different strategies in order to achive different goals; the available modes are the following:
    a. **disabled**: no agile beaconing is active; beacon will be generated according to the infos provided in the APRS Page (i.e. static operation based on time). This is the normal operation mode for time based beaconing.
    b. **EventMode**: will send a new beacon when an "event" is detected, like a change od direction, or the different speed of the mobile, or the stop/start of movement. Suitable thresholds are set to optimize this mode as a "**detailed radio coverage discovery**" mechanism. Not to be used for normal operation.
    c. **MappingModeXXX**; intended for creating a "**mapping**" of the present area where the device is , by sending a **beacon based on the "distance"** the mobile moved; XXX can assume following values: UltraFine, Detailed, Course, UltraCourse. Also this modes are not intended for normal operation, but for "**radio coverage discovery**".

3) "**Beacon Black List**": this feature allows to exclude a set of geographical "**zones**" from sending beacons, mainly for privacy purposes or to mitigate any radio congestion problems in particular areas such as eg. a town centre. This feature is not jet supported by a GUI page at present (will be supported ASAP) ; BTW it is already possible to define a zone list by touching the SW BuildDefinion File and re-building the SW via the PlatformIO development environment. .

4) Support of the "syslog" protocol towards a remote server for the purpose of collecting, mainly in the testing and diagnostic phase, a series of information useful for tracing the behavior of a device; this function obviously requires that the device under test is able to reach the internet directly, e.g. via a local wifi network (e.g. via a mobile phone operating in hotspot mode) or via the ethernet interface if any.

5) **Support of MQTT protocol**: this feature allows to send/receive data to/from a remote server (broker) again via internet connectivity. This function can be used in two ways:
    a. in a first way, the system allows you **to perform remote operation on a LoRa device** connected on the internet. e.g. it is possible to access some functions remotely or even perform operations on the device (e.g. restart the device or trigger other ad hoc functions). It's worth to be noted that this mode does not need any intervention or interaction in the local internet access device (router).
    b. A second way of using the **MQTT protocol is the classic one used in IoT** applications, which allows local data to be sent/received to/from a remote servers such as e.g. measurement of temperature, humidity or other types of parameters.

6) Use of **front panel button for quick functions:**
    a. a **short push** will request to "Send a Beacon" manually
    b. a **double short push** will show on the local display a number of parameters related tho the device (i.e dev. identity, IP address, etc.)
    c. **keeping the key pressed**, different options are presented on the display in sequence at intervals of few seconds. By releasing the key , the function shown on the display will be performed. A trivial example is the "**reboot**": by pressing the key after few seconds a "REBOOT NOW" will show up on the display: by releasing the key a reboot will be triggered. The functions available are:
        i. **Turn ON the upstream connection** (Non Standalone)

   ii. **Turn OFF the upstream connection** (Standalone)
   iii. **Reboot** the device

d. Keeping the the **key pressed after power on for 5-10 seconds** will trigger a "**Reset to Factory Defaults**" for the device to recover a clean device configuration. In addition, on Sarimesh HW devices, when the device is switched on, after few seconds the user can see the permanent switching ON of the green and red LEDs, during which a pressure of the key on the device is interpreted as a request to reset to the factory defaults, making this operation easyer.

7) Introduction of "**factory defaults**": after this function has been activated, the device will come up and operate in **APRS tracker mode with standard parameters** which have to be changed immediately for concrete use, but **they immediately allow you to have a device capable of functioning** giving signs of life and enabling the user to set his own personal parameters directly from the GUI interface accessible for example via a mobile phone or a PC. In particular **the device will comeup in "standalone"** operation mode: in this mode the device will operate as a WiFi access point (if WiFi interface is used) or can be connected via an Ethernet LAN cable (if Ethernet interface is provided) with DHCP server enabled so that the PC or control device will immediately and automatically acquire correct addressing to access the device; by using a smart-phone for this operation (with WiFi obviously), by connecting to the ESP-XXXX WiFi network created by the device, a browser window will automagically comeup for device operation.
If the SW in question is installed on HW devices other than the Sarimesh one, obviously the presence or absence of a small key and some LEDs must be taken into account.
In the case of TTGO type cards it is easy as there is a small key and a blue or red led which is mapped to operate as both the two red and green leds of the HW Sarimesh.
In the case of HELTEC boards that do not have a key functionally intended for a use other than a simple reset, unfortunately it will be necessary to restore the factory defaults by pushing bliendly the key for 5-10 seconds after device power-up.
A side effect of this feature in case of the TTGO like devices is that there is no need for loading anything in the LittleFS partition of the flash, thus simplifing the initial setup of the device.

8) Introduction of **Ethernet HW interface**: this is a very valuable function when using the devices in a remote site or when a very stable upstream internet connection is required, due to the fact that the WiFi interface as an upstream interface is often problematic in terms of automatic initialization or also in normal operation due to the presence of other WiFi networks in use locally. The new DM PCB will allow to implement either the WiFi interface or the Ethernet internet interface by installing suitable addons modules on the same PCB. In order to support the two HW variants a suitable matching SW image will be required.

9) Introduction of **Full Dual-Mode Dual-Radio LoRa interface**: this feature consists in installing on the same PCB two LoRa modules via their standard daugther boards, in order to have two cooperating LoRa radios able to implement the Dual-Mode radio strategy. **This feature requires mandatory use of second generation SX126x based LoRa modules**; the two modules will operate as follow
   a. **primary module will operate as both RX and TX** on a LoRa mode with a low SF (Spreading factor) to implement **hspeed LoRa mode**
   b. **secondary module will operate only as RX** for the legacy LoRa mode with high SF to implement the **lspeed legacy LoRa mode**

10) Introduction of **SW support for ambient sensors;** already on the old Sarimesh PCB there was support for 3 sensors; with new HW DM PCB the SW support has

been implemented for these sensors in order to send their measured values via APRS and/or via MQTT to upstream portals. In particular support has bee implemetd for following sensors:

   a. **BM680 temperature, pressure, humidity and VOC for CO2 and air quality index** measurement
   b. **BM280/BMP280 for simplified temperature, pressure and humidity measurement**
   c. **SPS30 Particulate Matter measurement** for 0.5, 1, 2.5 , 4 ad 10 nanom size

The on-board sensors can be optionally enabled/disabled as well as the **sending** of their **measurements via APRS**; in any case the acquired **measurements will be locally available via a suitable extension of the GUI, accessable via internet or direct WiFi/LAN interface**.

11) Introduction of **SW support for sending "working conditions infos"** via custom field **in the APRS comment**: this allows to record in the actual APRS packet the working conditions in use for the sending node for statistic purposes; this feature can be enabled/disabled via GUI either for every packet or for a packet time-to-time. The extension will include following infos (ex. 192B125S12C5P10):
   a. sequence number modulo 256
   b. Bxxx where xxx will be the Bandwidth in use
   c. Syy yy will be the Spreding Factor in use
   d. Cr r will be the coding rate in use
   e. Pss ss will be the preamble length in use

The objective of this info is to be able to **detect duplicate/missed packets** just in case.

12) **Reworking of the GUI and device configuration method**: till SW ver. 4.x the device configuration was performed via a GUI interface that was managing and modifying a number of parameters internally keept in a fast FRAM (for Sarimesh HW devices) and in slow and weareable flash (via a Little FS Filesystem) for TTGO T-Beam and similar devices. This was a major difference between Sarimesh and non Sarimesh HW devices and was implying a frequent access to the on processor flash memory that would slow the GUI operation and wear the flash for non Sarimesh devices. With SW Vr. 5.x there is now an improvement of the configuration method that should solve the slowing and frequent access to the flash and give GUI speed for the TTGO similar to the Sarimesh HW that incorporate a very fast FRAM . In pratice any configuration session for a device may include the access to few pages of the GUI that modify a number of parameters; in order to make a complete configuration several pages may be required to be modified; all these modifications will only impact an "in RAM memory" status of a "temporary configuration file" in JSON format that keeps normally the running device configuration. Any change to this "temporary configuration" will have immediate effect for the few "immediate action" parametrs (as for ex. the debug parameters) but will not be reflected in the "permanent configuration" keept in non volatile memory; in order to freeze this temporary configuration in the nonvolatile memory a "commit" operation will be required: this explicit operation will allow to set the present temporary configuration as the running configuration after the next device reboot. This method allows to discard , by not committing , any partial configuration change should the user decide to abort the re-configuration session; a reboot, without a previous committ, will restore the configuration to the last committed state. The commit actually will just copy the "in RAM memory" JSON configuration file to the FRAM or to the flash (depending on the HW device) in a one-shot operation; all the intermediate configuration changes will not impact the flash in any case.

13) Change/save of device configuration via Configuration file upload/download no longer requires to go trought the “Admin mode” step: till SW vr. 4.x this operations was requiring to put the device in a special “Admin status” and then would imply a double reboot for the device in order to save or restore a configuration file; now this step is no longer required ad a configuration can be saved or restored on/from file immediately; for restore operation from a configuration file, after the file upload a commit operation is required to make the new configuration the active one, after the next reboot.

14) The **GUI has been restructured for the APRS/LoRa parts** in order to support the new **Dual-Mode Dual-Radio feature** and make it more evident in usage; in particular an “APRS Features” page has been created holding the various attributes of APRS operation mode, including the enable/disable options for the various payload content components (like weather data, etc.). The LoRa page has been extended including different panels for the two RX and the single TX available depending on HW device; this allow to easyly configure the various operational modes also for a single radio device, just specifying different values for the RX and TX parameters (they can be set to completely different modes for split RX/TX operation).

15) New pages have been created to hold and display on board sensors date via a nice to see table in realtime.

16) The Dashboard page has been extended in order to contain explicit statistics for the two RX and the TX in order to have a detailed status of the device radios operation.

17) **A completely new feature at presentation level** has been introduced in addition to the already available APRS service and the Synchronous Beacon feature: it is the **LoRa Messenger application**: this is actually the “porting” of an existing piece of SW made available over Github by Nicholas (https://hackaday.io/project/166225-loramessenger) and implementing a simple “messaging application” operating directly at LoRa level, without passing trougth the classical APRS layer. It is a very light and interesting application that adds to the LoRa devices the capability to send and receive short text messages from other nodes reachable (only) at LoRa radio level; in addition it allows to have a list od recheable nodes at LoRa level; so **it is intended for “completely off-grid” operation and do no require or impact the internet connectivity**. The messaging strategy implemented by this application allow any node to operate as a relay for messages directed to other nodes and allow an “acknoledgment” operation to explicitely confirm a message reception toward the original message sender; both broadcast and directional messages are supported. For further details it is possible to check the Nicholas pages over the web. This application can operate in parallel with the APRS application and puts a light load on the actual radio channel; it can be enabled/disabled via the Operation Page of the device GUI. In order to use the LM Application a new GUI page has been created that just allow to read last received messages, write any new message and provide reachable (at LoRa level) active nodes list.

18) A new GUI page has been created for configuration of the Syslog server and the MQTT server previously hard coded in the SW.

19) Simplification of first SW installation method: now also the initial load module is a single file to be loaded to address 0x0000 of the flash, while up to vr. 4.x there was 5 different load modules to be loaded at different addressess . The single file also includes the LittleFS required for the TTGO HW devices. This makes very easy the setup of the flashing tool required for the first SW installation on a new device.

20) Local display layout has been sligthly changed in order to allow also with OLED display to have exactly the same content of the bigger color display. A new screen has been introduced containing the main device operation parameters like device name, IP address, GPS fix, date/time, etc. This screen is displayed after a two short pushes of the front button.

21) The buzzer , if installed on the DM PCB, can now be enabled/disabled via GUI.

22) A new section has been added in this user manual to describe the typical device setup suggested for integration of the new Dual-Mode devices in an old legacy LoRa network, to gain the benefit of the new mode also for devices not using the Sarimesh SW.

23) Old Sarimesh Devices was requiring some jumper setting to support different LoRa daughter boards due to different power supply required by the modules; new Sarimesh HW Devices are now completely jumper-less and the power supply is automatically adapted without any additional setting required on the HW. The daughterboard pinout has been extended to support the automatic power supply adaptation, but will maintain backward compatibility with the old format.

24) Old Sarimesh HW devices are still supported, while two new PCB are now available in order to have a smaller tracker including a LiPo battery and a dual radio PCB to support also the Ethernet interface and ambient sensors.

25) New PCB will now use extensively SMD components, so not easy to be soldered: to facilitate the DIY of these new PCB we will make available already semi-mounted PCB with the small SMD parts already mounted so that the device assembly can be easyly completed by only soldering Pin-In-Hole components without any special tool or experties beeing required. We are also evaluating an additional option of making available a full kit with all the required components in order to simplify the components procurement at a convenient price.

6 SARIMESH HW PCB variants available

Currently there are four HW variants of the LoRa_Beacon PCBs: the four variants differ only in the type of physical/functional plug-in modules supported and in the physical dimensions, but the basic SW functions are maintained in all the cases.

Following paragraphs will first document the two original (year 2020) HW devices, that will still be supported with the new SW vr. 5.x , and after that the new (year 2023) HW devices that will introduce new features and form factors .

Figure 1 summarize the different versions functional capabilities and major features.

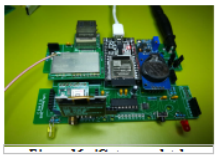
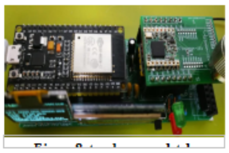
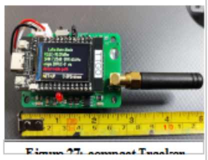
Feature	2020 iGate	2020 mini Tracker	2023 iGate DM	2023 compact Tracker
Device image				
HW Modularity	YES	YES	YES	NO
Processor Module	ESP32 Dev WROOM-32D 38pins	ESP32 Dev WROOM-32D 38pins	ESP32 Dev WROOM-32D 38pins	ESP32 TTGO -T-Display
Display	OLED 0,96" / TFT Color 1,14"	OLED 0,96" / TFT Color 1,14"	OLED 0,96" / TFT Color 1,14"	TFT Color 1,14"
Dual Display	YES	optional	YES	NO
RealTimeClock RTC	YES	NO	YES	NO
FRAM	YES	YES	YES	YES
GPS	Neo6 / Neo7 / Neo8	Neo6 / Neo7 / Neo8	ATGM336H	ATGM336H
External GPS Antenna	Optional	Optional	Optional	Optional
LoRa Modules	SX127x / SX126x	SX127x / SX126x	SX126x	SX1268
LoRa Dual-Radio	NO	NO	YES	NO
External LoRa antenna	SMA	SMA	SMA	SMA
Ethernet Interface	YES (no SW support)	NO	YES	NO
Buzzer	YES	YES	YES	NO
BME680/BMP680 Sensors	YES	YES	YES	NO
SPS30 Sensor	NO	NO	NO	YES
red/green leds	YES	YES	YES	YES (virtual)
Front Panel Button	YES	YES	YES	YES
LiPo Battery	NO	NO	NO	YES
Power supply	12V DC	5V USB-A	12V DC	5V USB-C
Mounting Technology	Pin-in-hole	Pin-in-hole	SMD	SMD

Figure 1 - Sarimesh LoRa PCB versions

6.1 LoRa_Beacon_2020_vr4_1 version (alias mini-Tracker)

The following 0 represent the wiring diagram and the Tracker PCB, which is characterized by being **very small in size and intended for mobile use** .

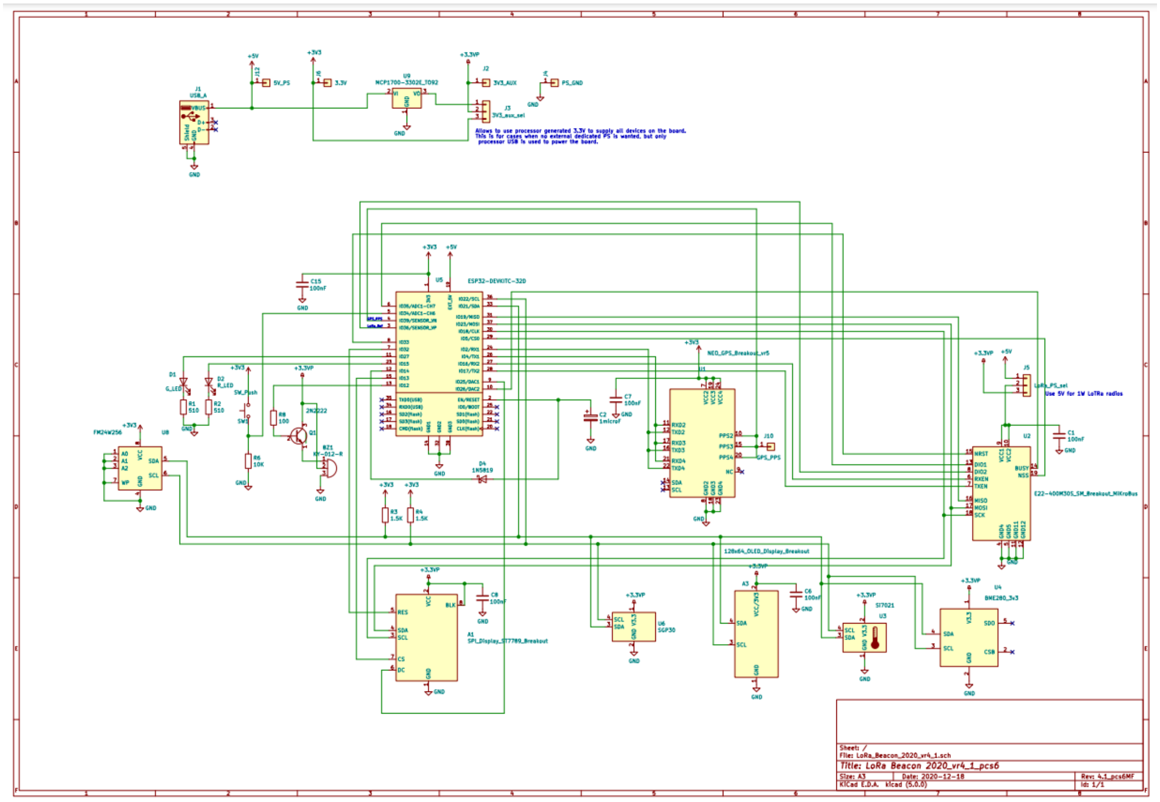


Figure 2 : LoRa_Beacon_2020_vr4.1_pcs6 version schematic diagram

This PCB (0) is designed to be powered via a USB Type "A" cable (therefore at 5V) using common sources such as a mobile phone power supply, a car USB socket or any power bank designed to power a classic mobile phone.

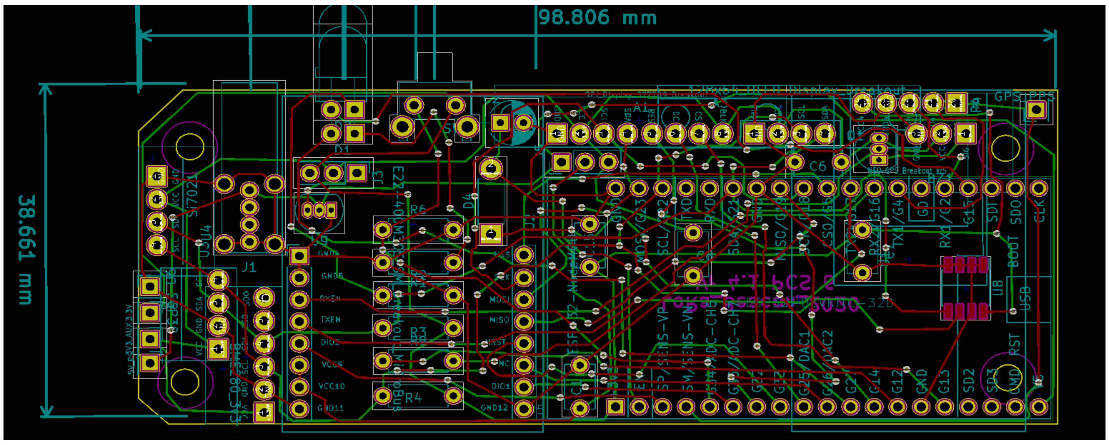


Figure 3 : LoRa_beacon_2020_Vr_4.1_pcs6 layout

It can operate as either an APRS Tracker or as an iGate; due to the dimension anyway this PCB is more suitable for mobile application.

The PCB keep the complete interchangeability of the common modules like GPS and LoRa with the other Sarimesh HW devices.

```
1       A1 - SPI_Display_ST7789_Breakout : APRS_Mini_Tracker:SPI_Display_ST7789_breakout
2       A3 - 128x64_OLED_Display_Breakout : APRS_Mini_Tracker:128x64_OLED_Vert_Display_breakout
3      BZ1 -          KY-012-R : APRS_Mini_Tracker:KY-012-R_breakout
4       C1 -             100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
5       C2 -           1microF : Capacitors_THT:CP_Radial_D5.0mm_P2.50mm
6       C6 -             100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
7       C7 -             100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
8       C8 -             100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
9      C15 -             100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
10      D1 -             G_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
11      D2 -             R_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
12      D4 -            1N5819 : Diodes_THT:D_DO-41_SOD81_P7.62mm_Horizontal
13      J1 -             USB_A : USB_A:USB_A_Vertical
14      J2 -           3V3_AUX : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
15      J3 -       3V3_aux_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
16      J4 -            PS_GND : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
17      J5 -       LoRa_PS_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
18      J6 -              3.3V : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
19     J10 -           GPS_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
20     J12 -             5V_PS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
21      Q1 -            2N2222 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
22      R1 -               510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
23      R2 -               510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
24      R3 -              1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
25      R4 -              1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
26      R6 -               10K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
27      R8 -               100 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
28     SW1 -           SW_Push : APRS_Mini_Tracker:Button_Angled
29      U1 - NEO_GPS_Breakout_vr5 : APRS_Mini_Tracker:NEO_GPS_Breakout_vr7
30      U2 - E22-400M30S_SM_Breakout_MiKroBus : APRS_Mini_Tracker:E22-M40030S_SM3_MiKroBus_Breakout_STACKED
31      U3 -            Si7021 : APRS_Mini_Tracker:SI-7021_breakout
32      U4 -        BME280_3v3 : APRS_Mini_Tracker:BME-280_breakout
33      U5 - ESP32-DEVKITC-32D : APRS_Mini_Tracker:MODULE_ESP32-DEVKITC-32D_STACKED
34      U6 -             SGP30 : APRS_Mini_Tracker:SGP30_breakout
35      U8 -          FM24W256 : Housings_SOIC:SOIC-8_3.9x4.9mm_Pitch1.27mm
36      U9 - MCP1700-3302E_TO92 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
```

Figure 4 : LoRa_Beacon_2020_vr_4.1_pcs6 part list

The part list is reported here in 0:

The 0 provides a more readable view of the placement of the various components on the PCB.

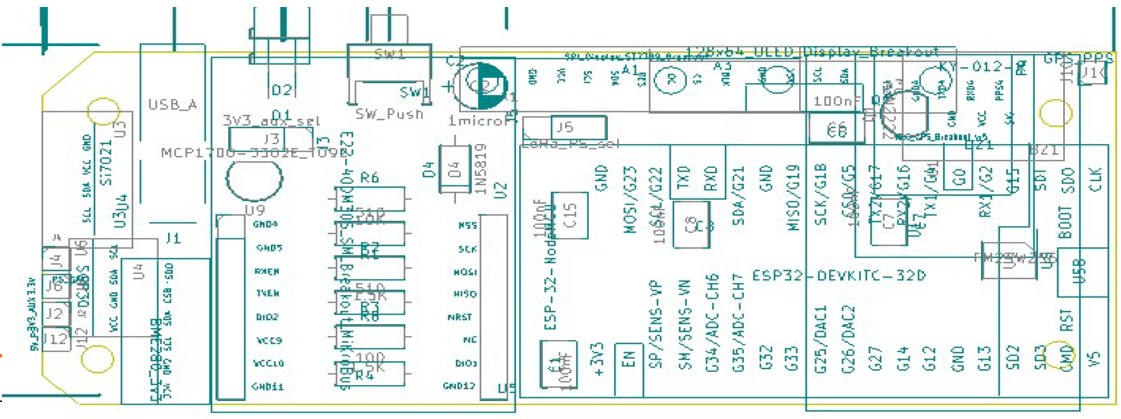


Figure 5 : LoRa Beacon Vr. 4.1-pcs6 : main components positioning

The following 0 provides a detail of the assembly of the single SMD chip present on the circuit; pin 1 corresponds to a small hole on the plastic body of the U8 device.

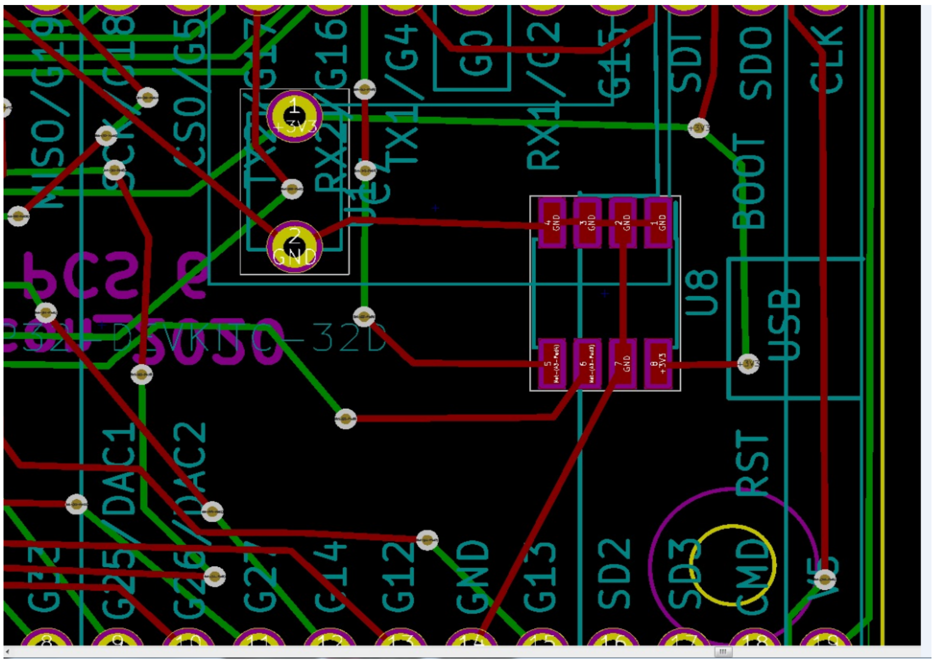


Figure 6: LoRa_Beacon Vr. 4.1_pcs6 : U8 SMD FRAM assembly detail

The following 0 shows a view of the component side of the Tracker PCB.

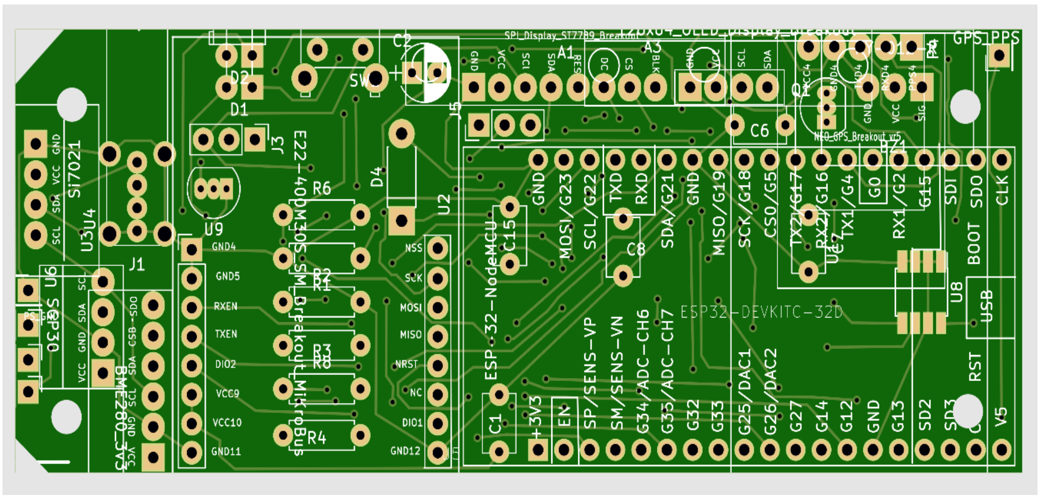


Figure 7:LoRa_Beacon Vr 4.1_pcs6 top side photo of the printed circuit board

0 shows a tracker completely assembled in one of its configuration

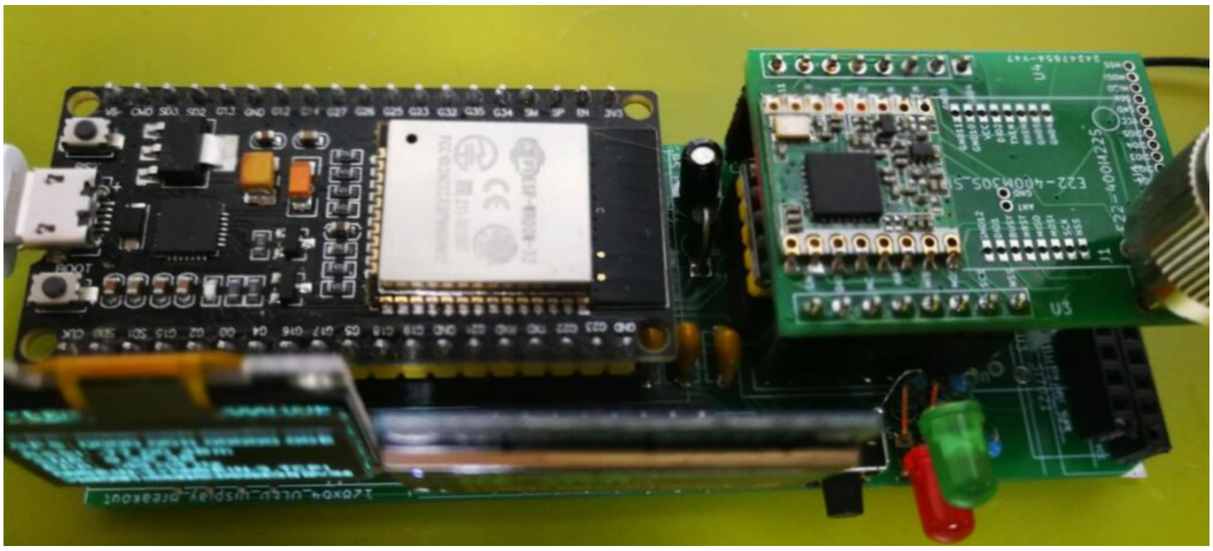


Figure 8: tracker completely assembled

0 is a side by side comparison of the mini Tracker and the TTGO-T-Beam device.

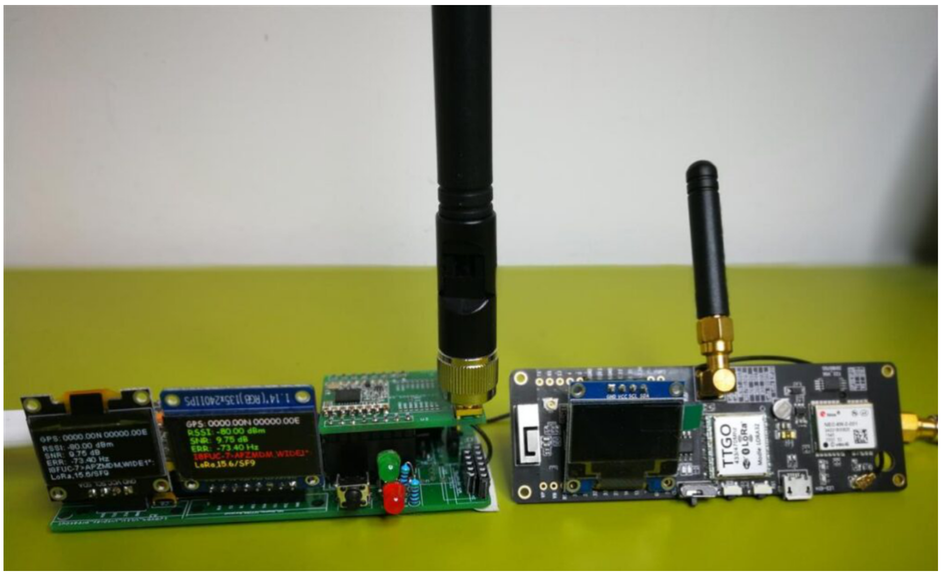


Figure 9 : Sarimesh HW compared to T-Beam

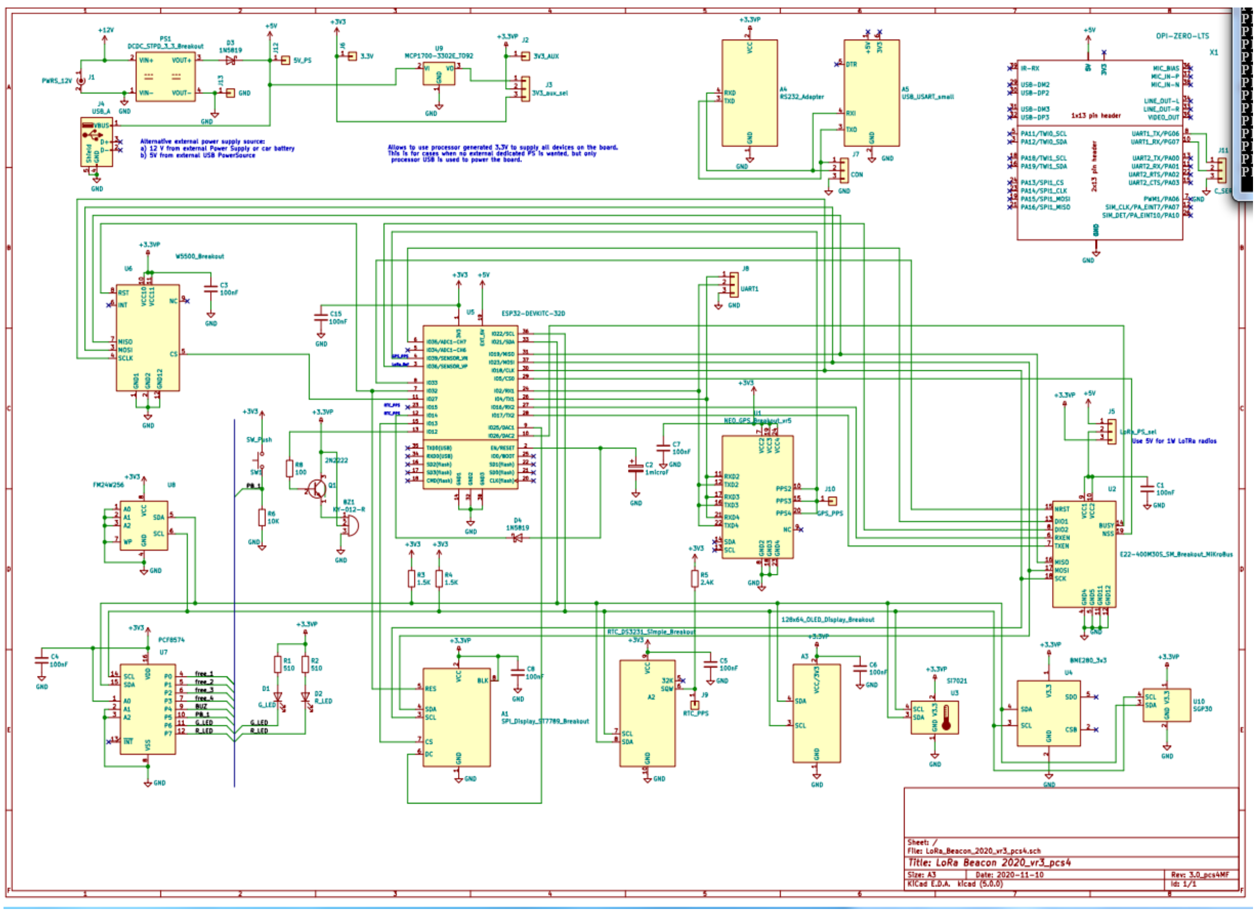
6.2 LoRa_Beacon_2020_vr3_pcs4 version (aka iGate)

The following figures represents the wiring diagram and the PCB of the iGate, which is characterized by being larger than the mini Tracker version and by having the provision for additional plug-in modules which allow more advanced experimentation compared to the mini Tracker.

Being this version designed to be used in a fixed site, a 12V power supply is foreseen.
Unlike the tracker application, the iGate has the possibility of connecting to the internet both in WiFi mode and via a LAN interface, if a dedicated plug-in module is installed. Moreover, it can hold an RTC (Real Time Clock) module for keeping local time even in the absence of a connection to a GPS device or the presence of internet connectivity.

It can operate as either an APRS Tracker or as an iGate; due to the dimension anyway this PCB is more suitable for fixed application.
The PCB keep the complete interchangeability of the common modules like GPS and LoRa with the other Sarimesh HW devices.

In the electrical diagram (0), the similarities are obviously very high even if the numbering and naming of the various components are different.



DCDC_STPD_3_3_Breakout
MCP1700-3302E_TO92
OPI-ZERO-LTS
W5500_Breakout
ESP32-DEVKITC-32D
NEO_GPS_Breakout_vr5
FM24W256
PCF8574
SPI_Display_ST7789_Breakout
RTC_DS3231_Simple_Breakout
128x64_OLED_Display_Breakout
E22-400M30S_SM_Breakout_MiKroBus
BME280_3x3
SI7021
SGP30
RS232_Adapter
USB_USART_small
Alternative external power supply source: a) 12 V from external Power Supply or car battery b) 5V from external USB PowerSource
Allows to use processor generated 3.3V to supply all devices on the board. This is for cases when no external dedicated PS is wanted, but only processor USB is used to power the board.
Use 5V for 1W LoRa radios
File: LoRa_Beacon_2020_vr3_pcs4.sch
Title: LoRa Beacon 2020_vr3_pcs4
Size: A3 Date: 2020-11-10 Rev: 3.0_pcs4MF
KiCad E.D.A. kicad (5.0.0) Id: 1/1

Figure 10 : LoRa_Beacon_2020_vr3_pcs4 : wiring diagram

Details of this version follow. 0 is the PCB layout.

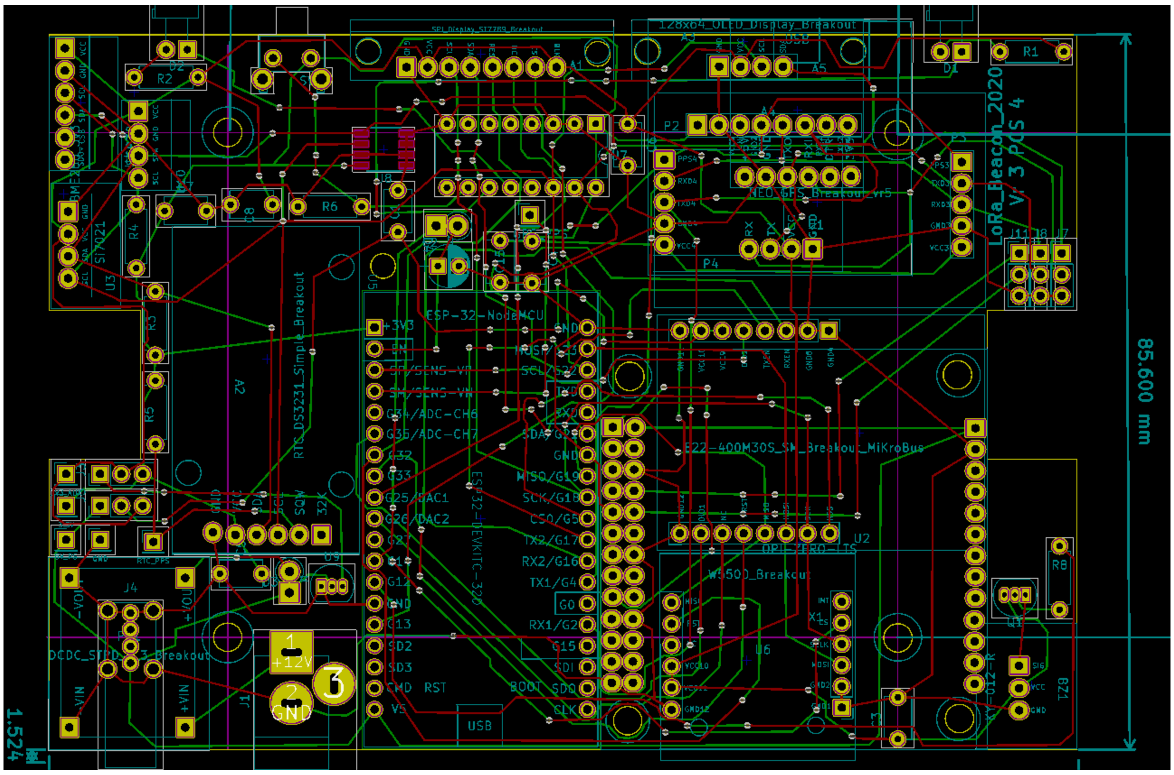


Figure 11: LoRa_Beacon_2020_vr3_pcs4 : Carrier PCB layout

Here below, the part list (0) of the iGate version is reported :

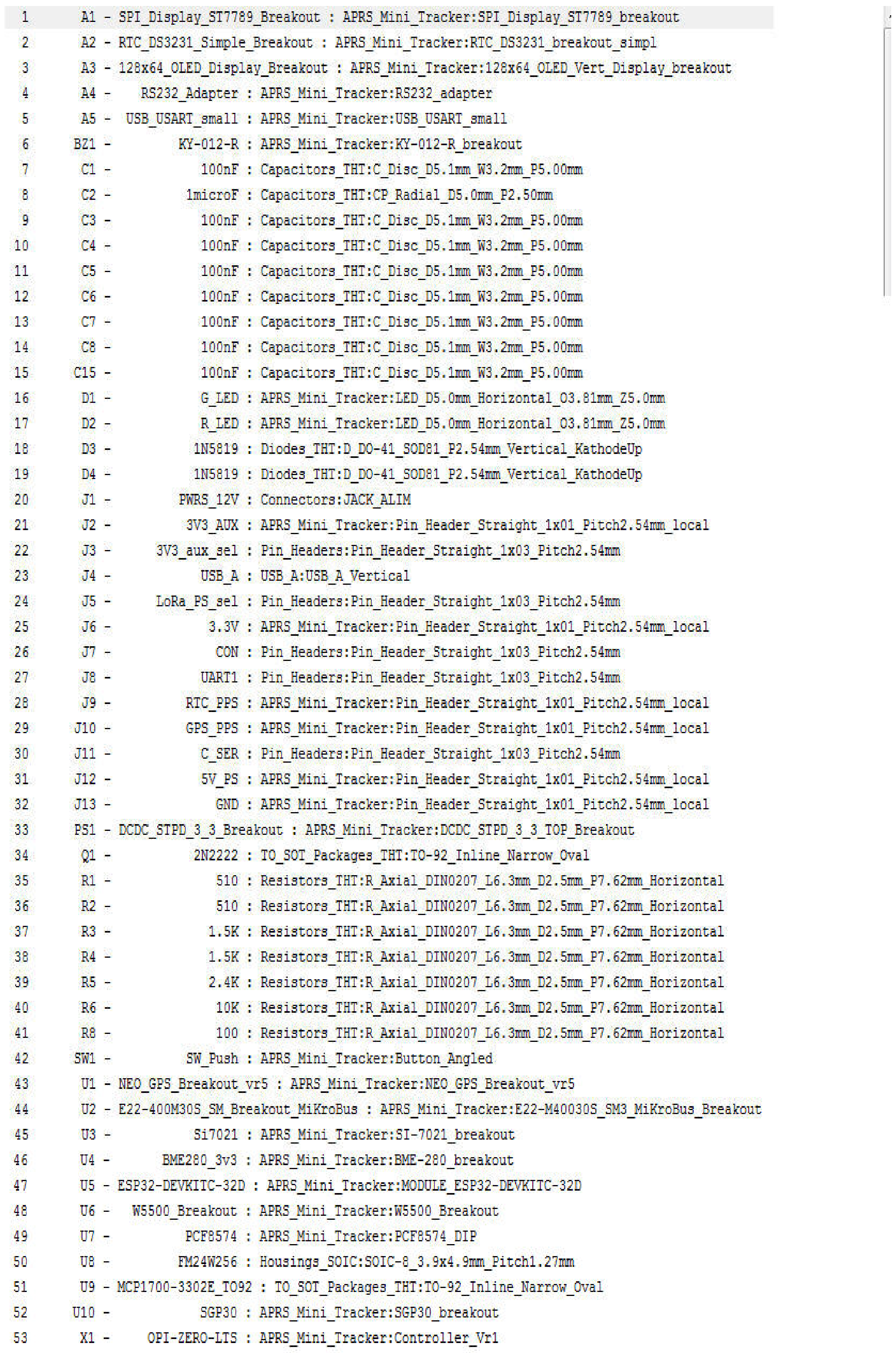

```
 A1 - SPI_Display_ST7789_Breakout : APRS_Mini_Tracker:SPI_Display_ST7789_breakout
 A2 - RTC_DS3231_Simple_Breakout : APRS_Mini_Tracker:RTC_DS3231_breakout_simpl
 A3 - 128x64_OLED_Display_Breakout : APRS_Mini_Tracker:128x64_OLED_Vert_Display_breakout
 A4 -    RS232_Adapter : APRS_Mini_Tracker:RS232_adapter
 A5 -  USB_USART_small : APRS_Mini_Tracker:USB_USART_small
BZ1 -         KY-012-R : APRS_Mini_Tracker:KY-012-R_breakout
 C1 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C2 -          1microF : Capacitors_THT:CP_Radial_D5.0mm_P2.50mm
 C3 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C4 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C5 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C6 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C7 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 C8 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
C15 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 D1 -            G_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
 D2 -            R_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
 D3 -           1N5819 : Diodes_THT:D_DO-41_SOD81_P2.54mm_Vertical_KathodeUp
 D4 -           1N5819 : Diodes_THT:D_DO-41_SOD81_P2.54mm_Vertical_KathodeUp
 J1 -         PWRS_12V : Connectors:JACK_ALIM
 J2 -          3V3_AUX : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
 J3 -      3V3_aux_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
 J4 -            USB_A : USB_A:USB_A_Vertical
 J5 -      LoRa_PS_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
 J6 -             3.3V : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
 J7 -              CON : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
 J8 -            UART1 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
 J9 -          RTC_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
J10 -          GPS_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
J11 -            C_SER : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
J12 -            5V_PS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
J13 -              GND : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
PS1 - DCDC_STPD_3_3_Breakout : APRS_Mini_Tracker:DCDC_STPD_3_3_TOP_Breakout
 Q1 -           2N2222 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
 R1 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R2 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R3 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R4 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R5 -             2.4K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R6 -              10K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
 R8 -              100 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
SW1 -          SW_Push : APRS_Mini_Tracker:Button_Angled
 U1 - NEO_GPS_Breakout_vr5 : APRS_Mini_Tracker:NEO_GPS_Breakout_vr5
 U2 - E22-400M30S_SM_Breakout_MiKroBus : APRS_Mini_Tracker:E22-M40030S_SM3_MiKroBus_Breakout
 U3 -           Si7021 : APRS_Mini_Tracker:SI-7021_breakout
 U4 -       BME280_3v3 : APRS_Mini_Tracker:BME-280_breakout
 U5 - ESP32-DEVKITC-32D : APRS_Mini_Tracker:MODULE_ESP32-DEVKITC-32D
 U6 -   W5500_Breakout : APRS_Mini_Tracker:W5500_Breakout
 U7 -          PCF8574 : APRS_Mini_Tracker:PCF8574_DIP
 U8 -         FM24W256 : Housings_SOIC:SOIC-8_3.9x4.9mm_Pitch1.27mm
 U9 - MCP1700-3302E_TO92 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
U10 -            SGP30 : APRS_Mini_Tracker:SGP30_breakout
 X1 -     OPI-ZERO-LTS : APRS_Mini_Tracker:Controller_Vr1
```

Figure 12: LoRa_Beacon_2020_vr3_pcs4 : part list

Here below (0), there is a view of the placement of the components:

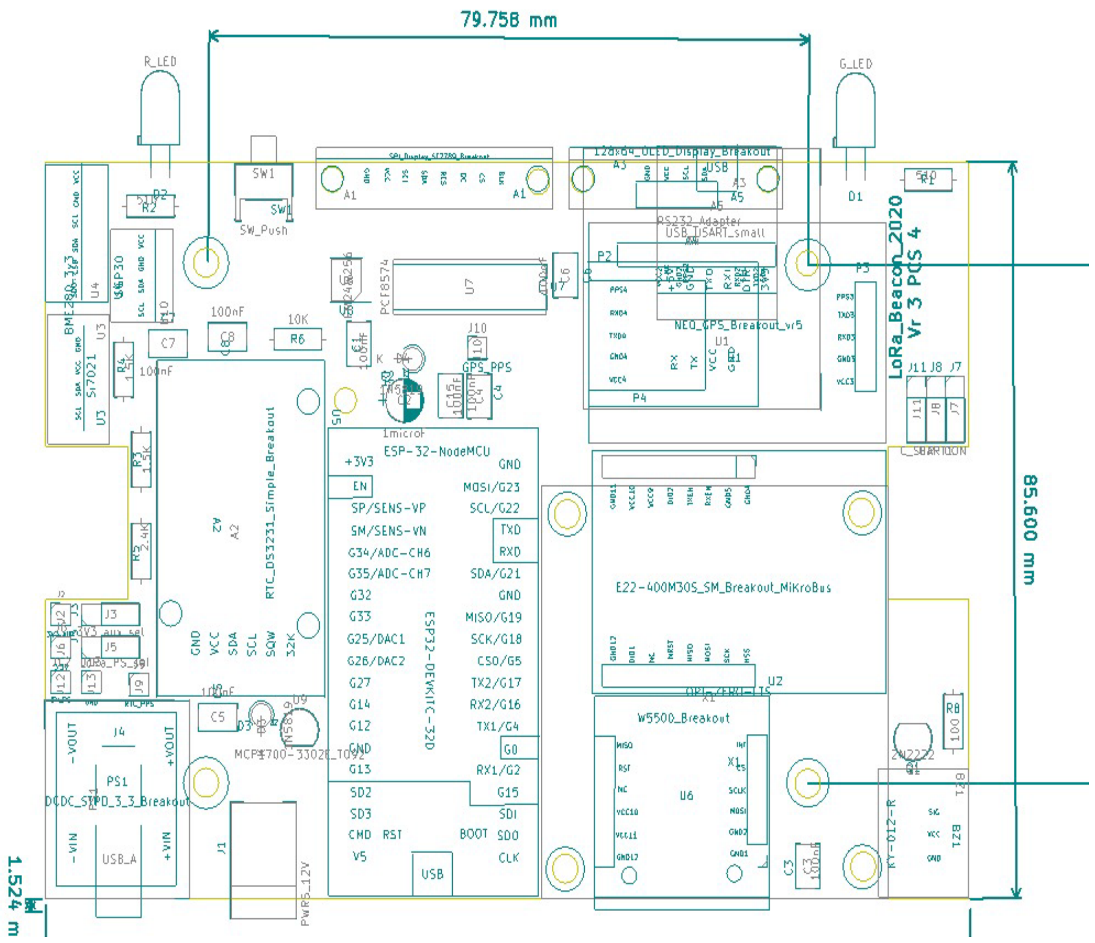


Figure 14: LoRa_Beacon_2020_vr3_pcs4 : main components placement

Here below (0), the assembly details of the single SMD U8 component are reported.

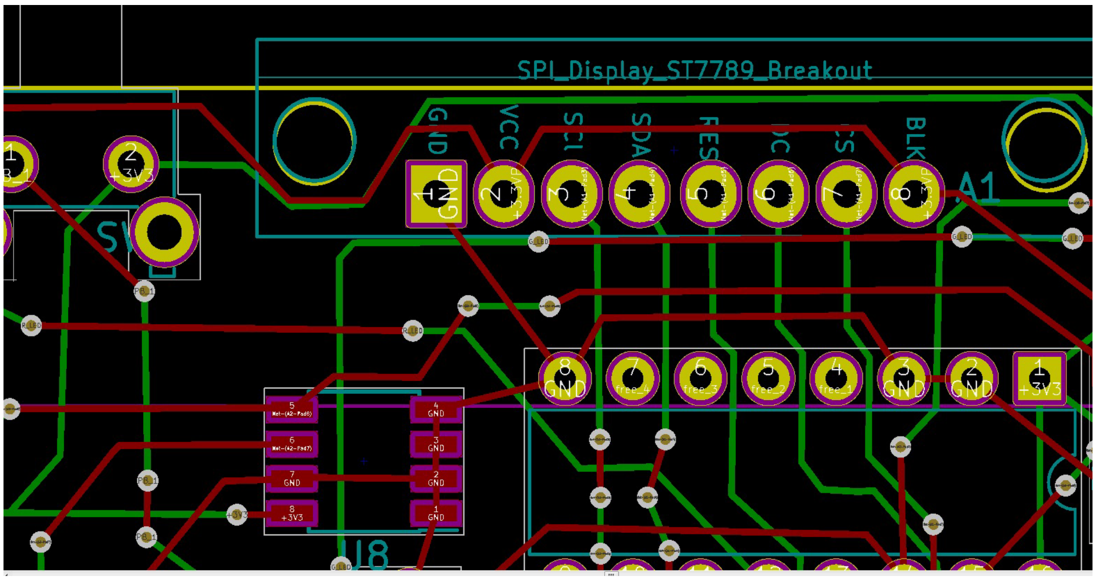


Figure 13: LoRa_Beacon_2020_vr3_pcs4 : U8 SMD FRAM positioning detail

The following 0 is a photo of the component side of the PCB of this version

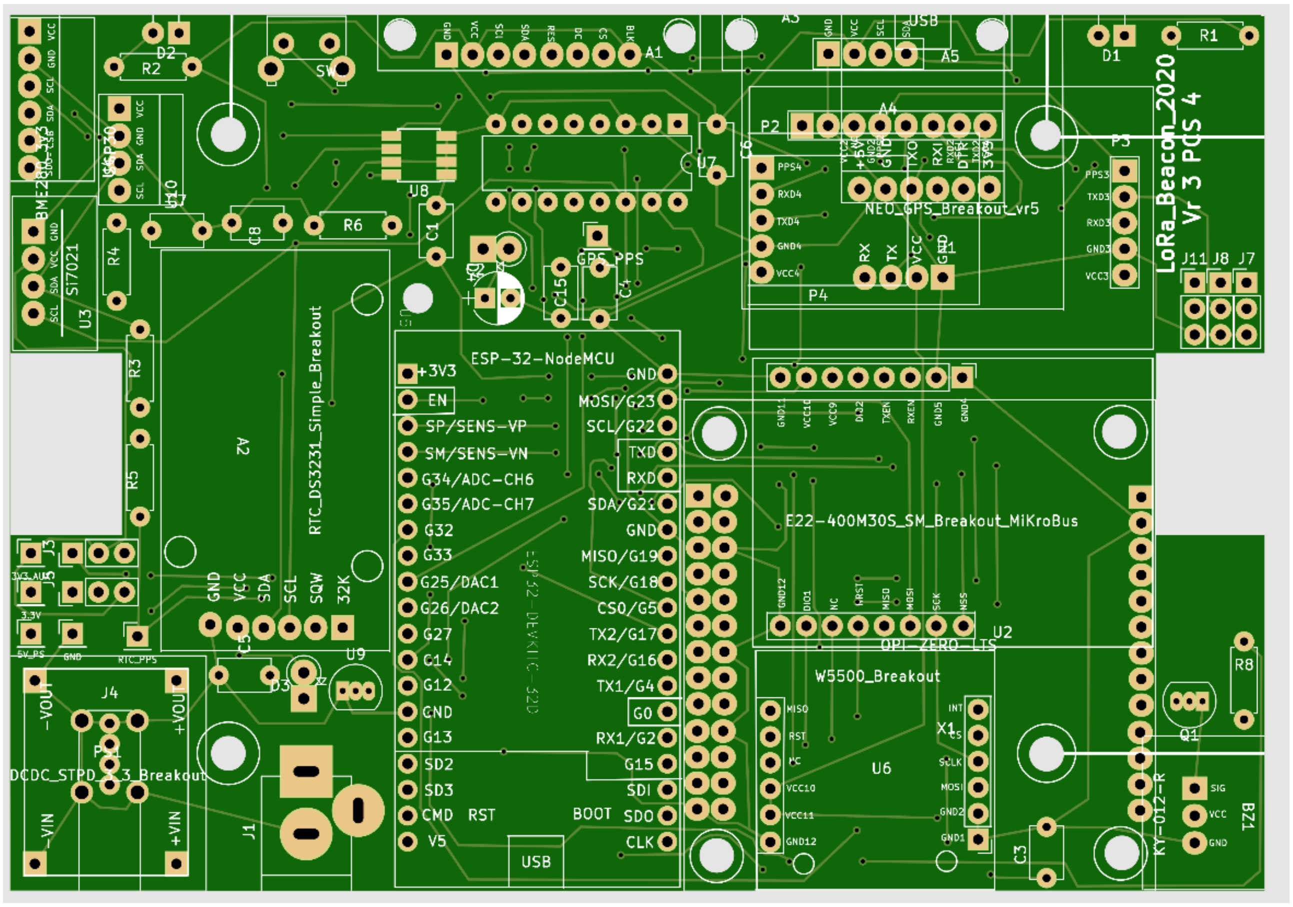


Figure 15: LoRa_Beacon_Vr 3 pcs 4 : PCB top side view

0 shows an iGate completely assembled in one of the possible configuration.

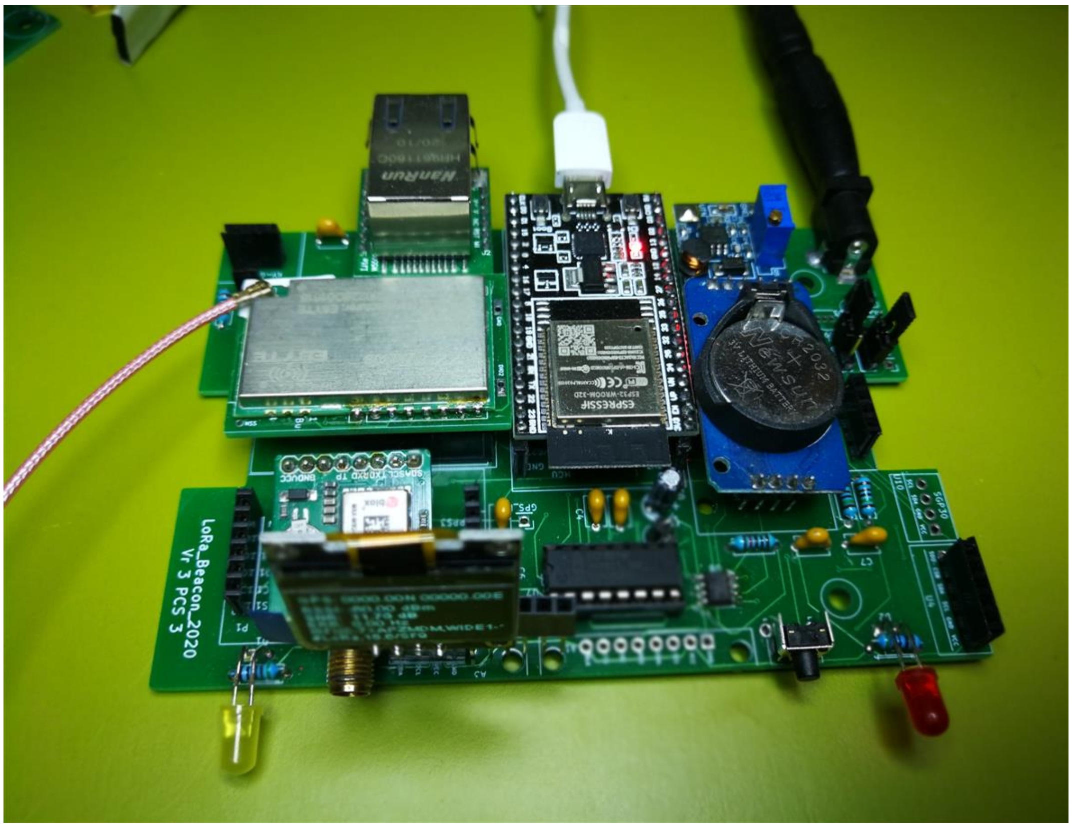


Figure 16 : iGate completely assembled

6.3 Carrier for LoRa radio modules (LoRa carrier)

Unfortunately, the LoRa radio modules available on the market have a considerable diversity of pin layout and physical format, not always compatible with a 2.54mm pin pitch that characterizes the majority of the other modules required for our implementation.

For these reasons, we designed a PCB with the aim to adapt the pinout of the various LoRa modules on the market to the standard 2.54mm receptacles present on the main carrier.

We have developed different carriers, but in this paragraph is reported the more useful, at least initially.

The following 0 represent the wiring diagram and the PCB of this LoRa carrier version:

As can be seen, the PCB has many different fingerprints:

- Ebyte E22-400M30S and similar
- Ebyte E22-400M22S and similar
- RFM98 and similar
- DRF1278F and similar

Obviously, only one of the fingerprints will be used at a time; to mount different LoRa chips, different samples of carrier + LoRa chips will therefore be built.

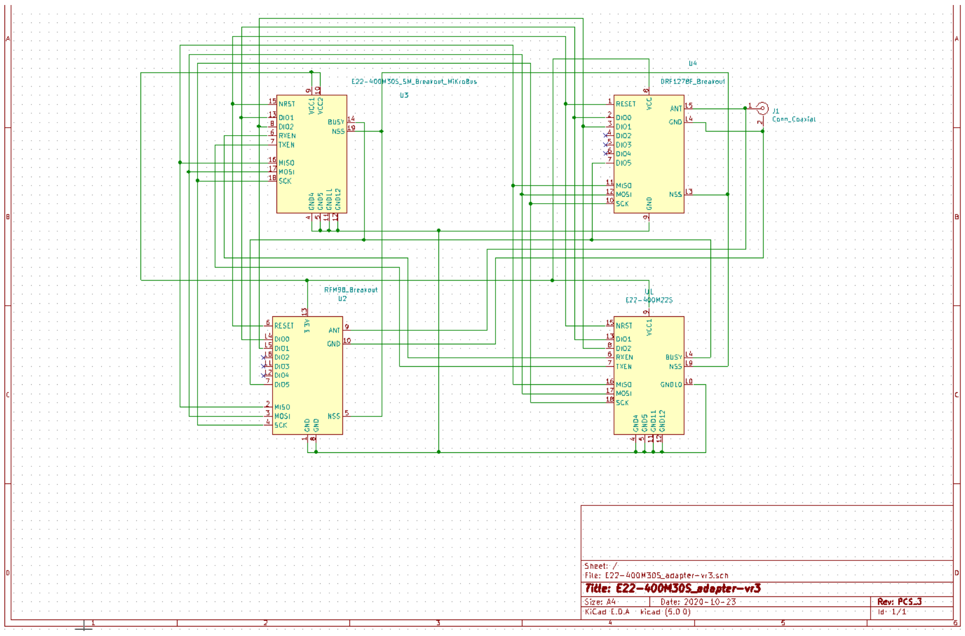


Figure 17 : LoRa carrier wiring diagram

The 0 and 0 below, a photo of the top side of the PCB is reported.

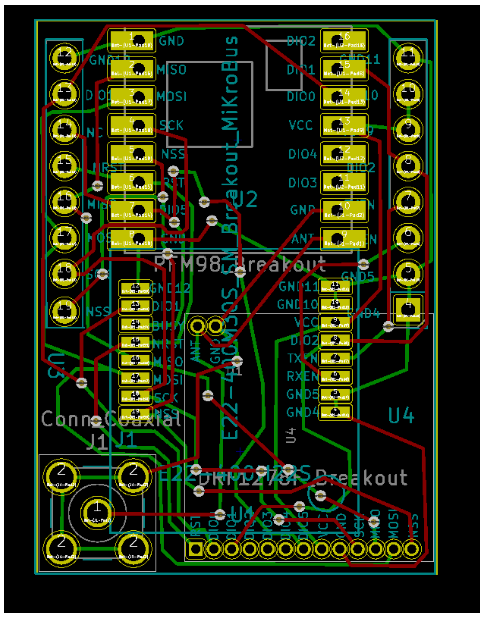


Figure 18 : LoRa carrier PCB layout

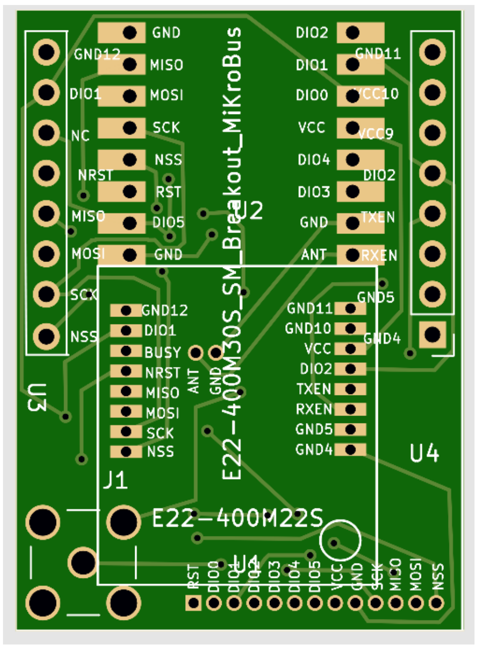


Figure 19: LoRa_Carrier PCB top

One of the most popular LoRa module is the E22-400M30S, which already has a pin spacing of 2.54 mm even if with an SMD mounting layout.

Also in this case, the carrier will allow the radio module to be mounted in an optimal way. The following figures illustrate how this module is mounted.

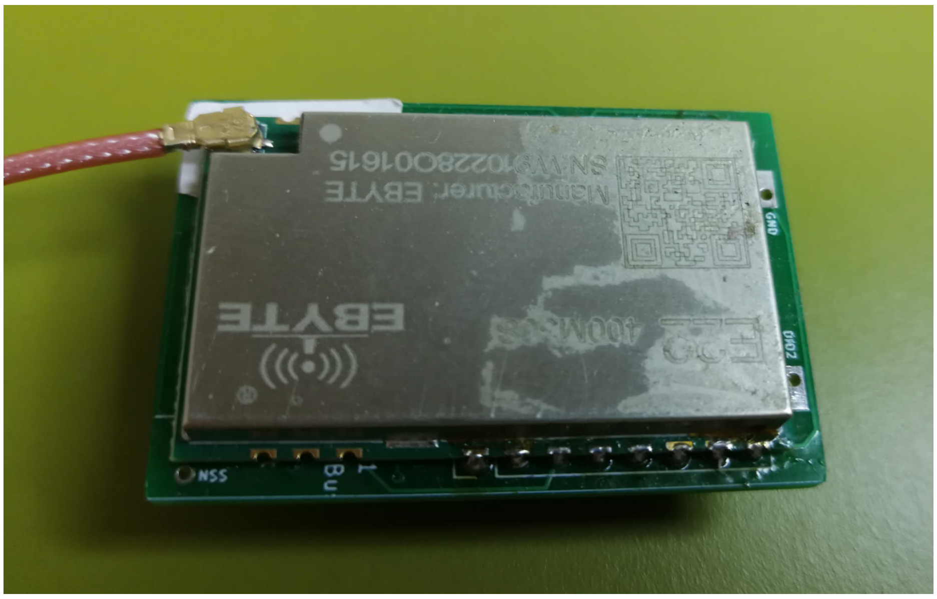


Figure 20: LoRa module mounted on LoRa carrier PCB – view1

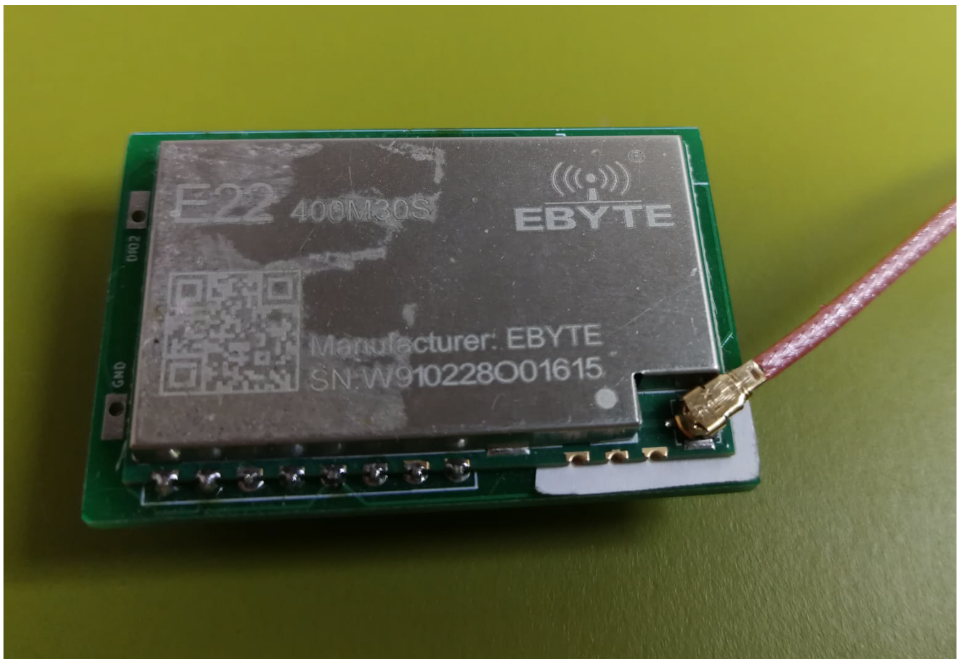


Figure 21: LoRa module mounted on LoRa carrier PCB – view2

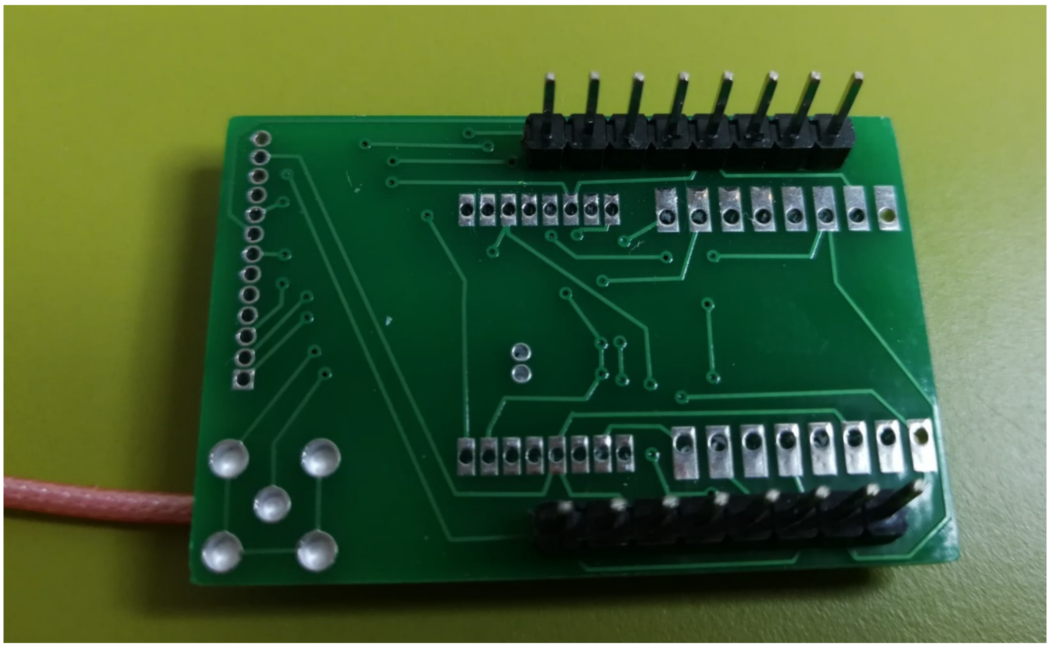


Figure 22: LoRa module mounted on LoRa carrier PCB - bottom view

6.4 Assembly notes related to both version 2020 PCB main board versions.

On both main board PCB year 2020 versions there are some provisions for optional plug-in modules.

The first is for the display modules: both versions have the footprints and connections compatible for both color and monochrome display, which can be used alternatively or even simultaneously through an appropriate customization of the SW; in the standard version of the SW, a color TFT module with SPI interface and an OLED module with I2C interface are simultaneously supported, however showing the same type of content.

A further note always concerns the color display module which, depending on the version purchased, may or may not have the pins already soldered on the relative PCB: for an optimal assembly it is necessary to use pins with a 90° bend; if straight pins are already soldered on the module, it is necessary to remove them and replace them with pins bent at 90°; to simplify the work of desoldering the original pins, it is better to cut the plastic support that binds the original pins with a pair of nippers before starting the desoldering process.

All the modules can be plugged on the Carrier/Mother Board using a row of standard 2.54mm pitch connectors.

This allows in the future to be able to easily replace or interchange the modules without complicated desoldering operations which would inevitably ruin the circuits.

For both PCB versions the sensor modules are optional and not supported in the base SW up to vr. 4.x; they are now supported with SW vr. 5.x.

The GPS module is mandatory for the tracker version while it is optional for the iGate version.

The RTC module for the iGate version is supported by SW but is still optional.

The GPS module has been provided on both carrier PCBs with a footprint that can be adapted to different types of modules available on the usual internet purchase portals; for further details on compatible modules, contact the writer by e-mail

Note: the GPS module must have PPS OUT pin.

The iGate version has two power options: 5V connection through a USB cable or via a 12V connection received from a jack connector; the default version uses the 12V power supply and therefore requires the PS1 DC/DC module and its power connector.

As a general note, it is recommended to test the circuit by gradually connecting the various modules, starting with the ESP32 processor and continuing with the display, the GPS and finally the LoRa module.

In order to possibly allow trobleshouting actions, simple "test segments" are available, i.e. small programs which, when suitably loaded, via the Arduino SW development environment or, in version 4.x, PlatformIO on the ESP32 processor can allow you to test the various functional blocks individually or in small groups.

A Troubleshooting guide is also available as appendix of this document.

Finally, there are some configuration that can be made using appropriate jumpers with 2.54mm format which are documented below:

6.4.1 Mini-tracker PCB version: jumpers and their meaning

- **J3**: 3V3_aux_sel default connect pins 1-2
- **J5**: LoRa_PS_sel use position 1-2 for LoRa module E22-400M30S operating at 5V; position 2-3 for LoRa modules working at 3.3V

6.4.2 iGate PCB version: jumpers and their meaning

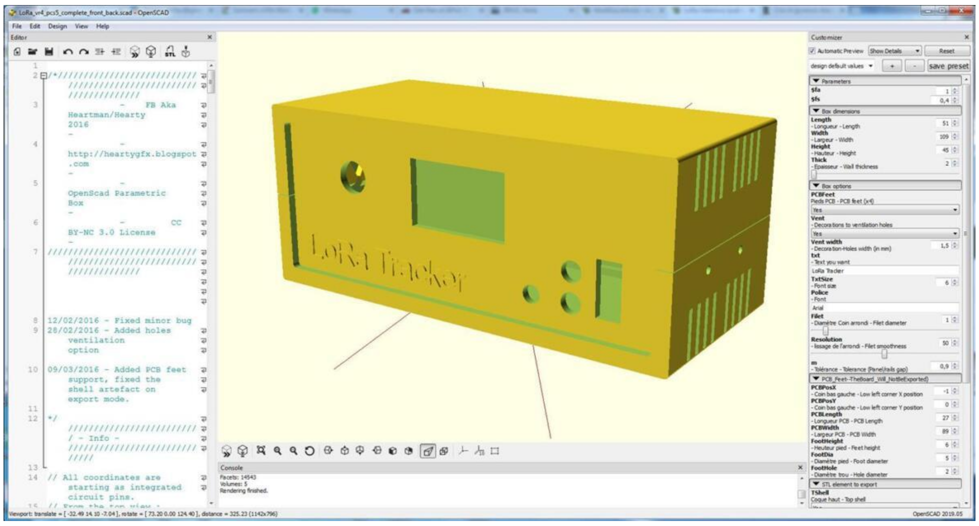
- **J3**: 3V3_aux_sel default connect pins 1-2
- **J5**: LoRa_PS_sel use position 1-2 for LoRa module E22-400M30S operating at 5V; position 2-3 for LoRa modules working at 3.3V
- **J7**: for future expansions, no jumpers to connect
- **J8**: for future expansions, no jumpers to connect
- **J11**: for future expansions, no jumper to connect

6.5 Plastic Enclosure for Mini Tracker 2020 version

3D case projects are available on Github repository for both Tracker and Igate.
They are a “baseline” and the user can easily customize the projects following his needs, or can adapt commercial boxes.

The source files can be opened by OpenScad, a freeware software freely downloadable from the internet: https://openscad.org/

Here below, a picture (0) of tracker case viewed in OpenScad environment, only for example.



LoRa_vr4_pcs5_complete_front_back.scad - OpenSCAD
File Edit Design View Help
Editor
Customizer
LoRa Tracker
OpenSCAD 2019.05

```
/*//////////////////////////////////////////////////////////////
              -    FB Aka Heartman/Hearty 2016                   -
              -    http://heartygfx.blogspot.com                 -
              -    OpenScad Parametric Box                       -
              -    CC BY-NC 3.0 License                          -
////////////////////////////////////////////////////////////////
12/02/2016 - Fixed minor bug
28/02/2016 - Added holes ventilation option
09/03/2016 - Added PCB feet support, fixed the shell artefact on export mode.

*/ ////////////////////////// - Info - //////////////////////////////

// All coordinates are starting as integrated circuit pins.
// From the top view :
```

```
Console
Facets: 14543
Volumes: 5
Rendering finished.
```

```
Viewport: translate = [ -32.49 14.10 -7.04 ], rotate = [ 73.20 0.00 124.40 ], distance = 325.23 (1142x796)
```

```
Automatic Preview [✓]   Show Details   Reset
design default values   +   -   save preset
Parameters
$fa                                                       1
$fs                                                       0,4
Box dimensions
Length  -Longueur - Length                                51
Width   -Largeur - Width                                  109
Height  -Hauteur - Height                                 45
Thick   -Epaisseur - Wall thickness                       2
Box options
PCBFeet  Pieds PCB - PCB feet (x4)                        Yes
Vent     -Decorations to ventilation holes                Yes
Vent width  -Decoration-Holes width (in mm)               1,5
txt      -Text you want                                   LoRa Tracker
TxtSize  -Font size                                       6
Police   -Font                                            Arial
Filet    -Diamètre Coin arrondi - Filet diameter          1
Resolution  -lissage de l'arrondi - Filet smoothness      50
m        -Tolérance - Tolerance (Panel/rails gap)         0,9
PCB_Feet--TheBoard_Will_NotBeExported)
PCBPosX  -Coin bas gauche - Low left corner X position    -1
PCBPosY  -Coin bas gauche - Low left corner Y position    0
PCBLength  -Longueur PCB - PCB Length                     27
PCBWidth   -Largeur PCB - PCB Width                       89
FootHeight -Heuteur pied - Feet height                    6
FootDia    -Diamètre pied - Foot diameter                 5
FootHole   -Diamètre trou - Hole diameter                 2
STL element to export
TShell   Coque haut - Top shell
```

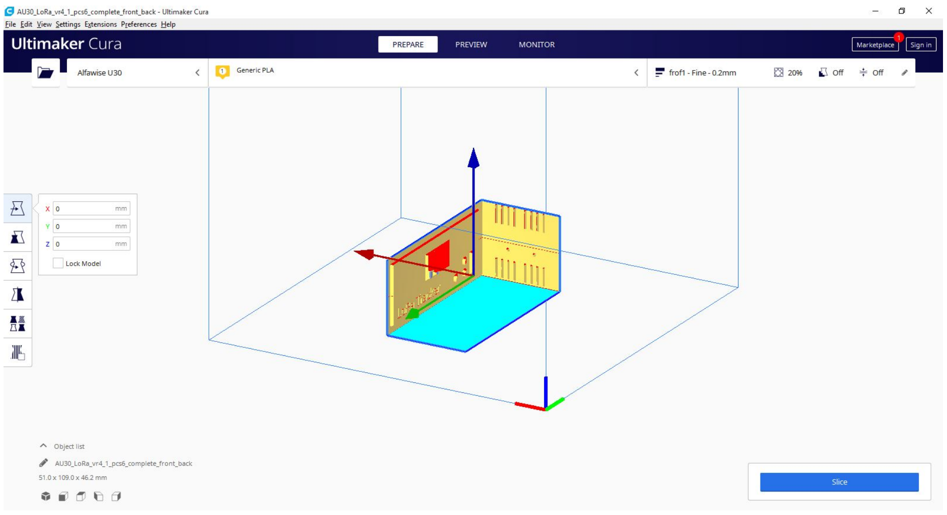
Figure 23: Tracker Case viewed in OpenSCAD

Note: the dimensions of the case are parametric, so that the case can be easily adapted to other projects. The correct Height, Length and Width are reported below:

- Length: 51
- Width: 109
- Height: 45

Once the project is closed in OpenScad, it can be viewed in "Ultimaker Cura", which is a free, easy-to-use 3D printing software downloadble here: https://ultimaker.com/software/ultimaker-cura/ . With this Software, it's possible to do final "slicing" and the generation of the file wich can be processed by the 3D printer.

Here below, a view of the same case in Ultimaker Cura is shown (0):



AU30_LoRa_vr4_1_pcs6_complete_front_back - Ultimaker Cura
File Edit View Settings Extensions Preferences Help
Ultimaker Cura
PREPARE
PREVIEW
MONITOR
Marketplace
Sign in
Lock Model
Object list
Slice

```
Alfawise U30    Generic PLA    frof1 - Fine - 0.2mm    20%    Off    Off
X 0 mm
Y 0 mm
Z 0 mm
AU30_LoRa_vr4_1_pcs6_complete_front_back
51.0 x 109.0 x 46.2 mm
```

Figure 24: Tracker Case viewed in Ultimaker Cura

6.6 Plastic Enclosure for iGate 2020 version

The PCB has been designed to fit exactly in a standard plastic case available at the following seller (https://secure.reichelt.com/it/it/kunststoffgehaeuse-serie-rm-100-x-130-x-50-mm-grau-rm2015m-p221333.html?&nbc=1); it is possible to adapt the front/back panel with some tooling beeing the plastics easyly workable.

6.7 LoRa_Compact_Tracker_2023_vr2 version (alias compact-Tracker)

This new PCB has been designed to be a very compact device to keep in the pocket for open air applications. It is a non modular device that is based on a standard TTGO T-Display processor module carried by a custom PCB that includes the specific modules required to implement a tracker/iGate device able to support the full set of Sarimesh SW Vr. 5.x functions in a very minimal weigth and space.
The main processor includes all the necessary ESP32 processor features and an 1.14" color display, plus the necessary circuits for LiPo battery namagement and charging.

The PCB carries a LoRa SX1268 second generation radio , a fully featured GPS , a FRAM and a LiPo battery with power switch .

The PCB is implement with SMD tecnology and is able to mount an external antenna via an SMA connector; a GPS antenna is internally conneted and, just in case, a suitable connector can be equipped in order to connect an external GPS antenna.

In the following 0 the schematics and the PCB images (0) are presented as well as the device BOM.

The design is completely jumper-less and includes a power switch to shutdown the device if required ; to recharge the internal LiPo battery a standard USB-C connector is provided; this single connector can also be used for initial SW loading and for monitoring and debug operations via a serial-over-usb connection to a PC.

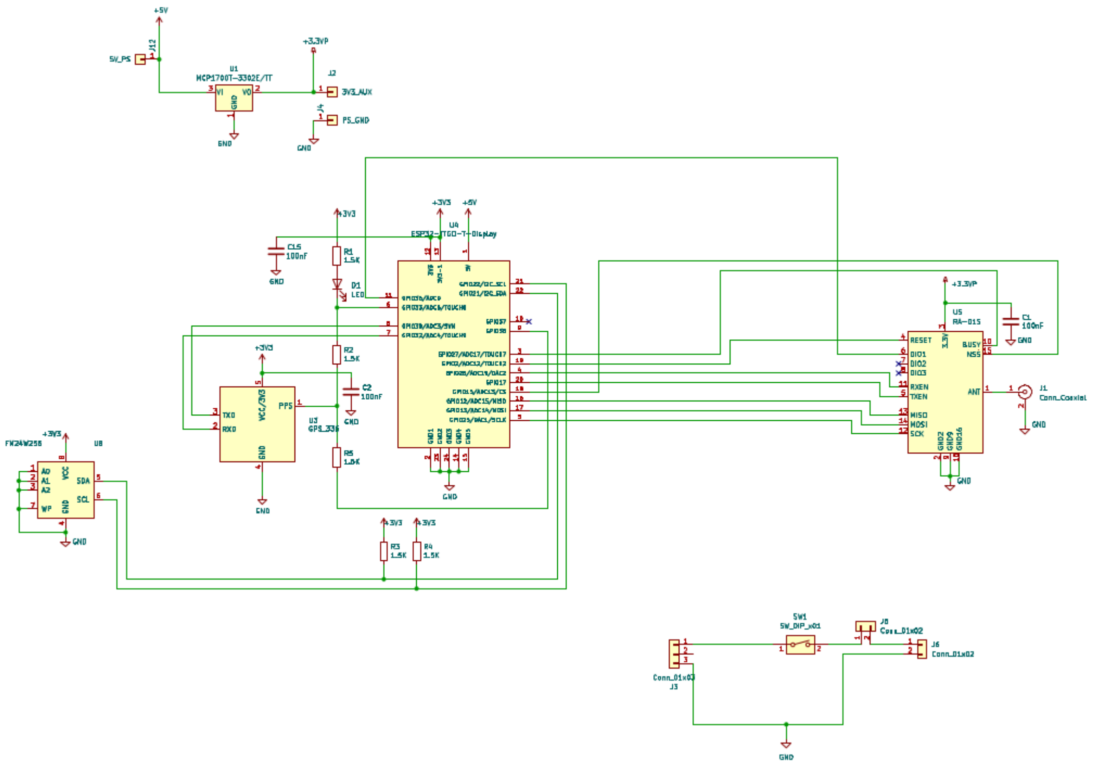


Figure 25: Compact Tracker 2023 schematics

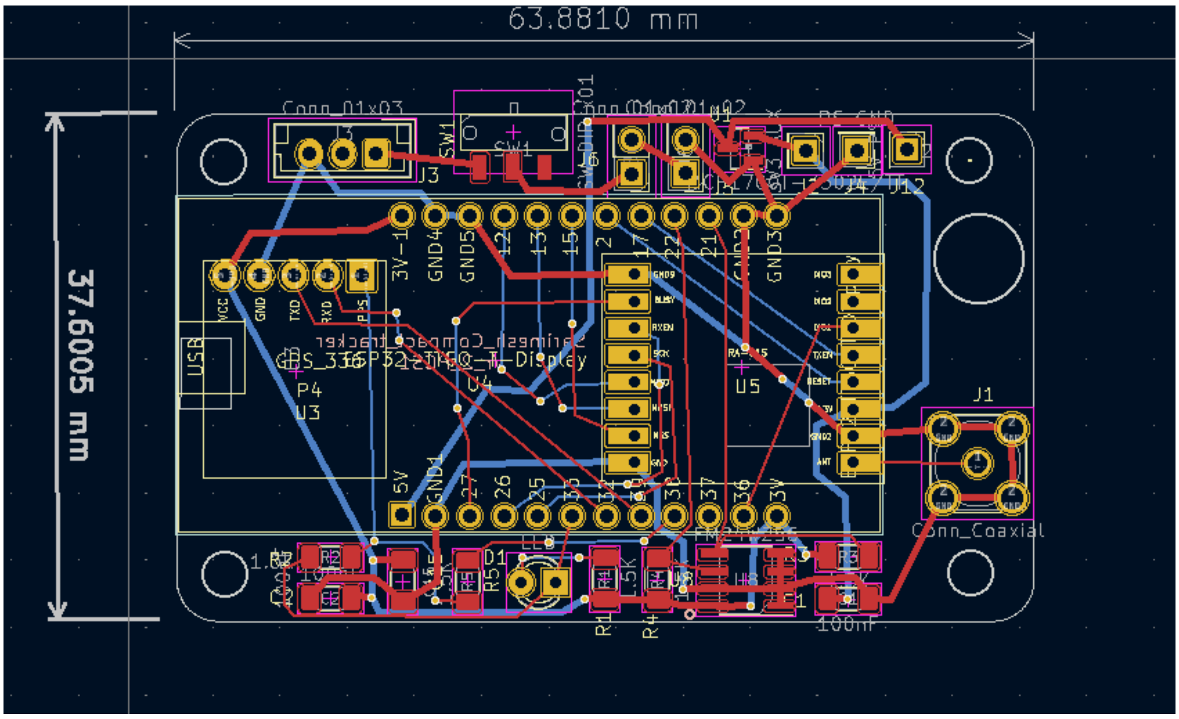


Figure 26: compact tracker PCB layout

The next image 0 is a sample ; the LiPo Battery is on the back side of the PCB and can be of different capacity; the suggested capacity is 1400 mAh.

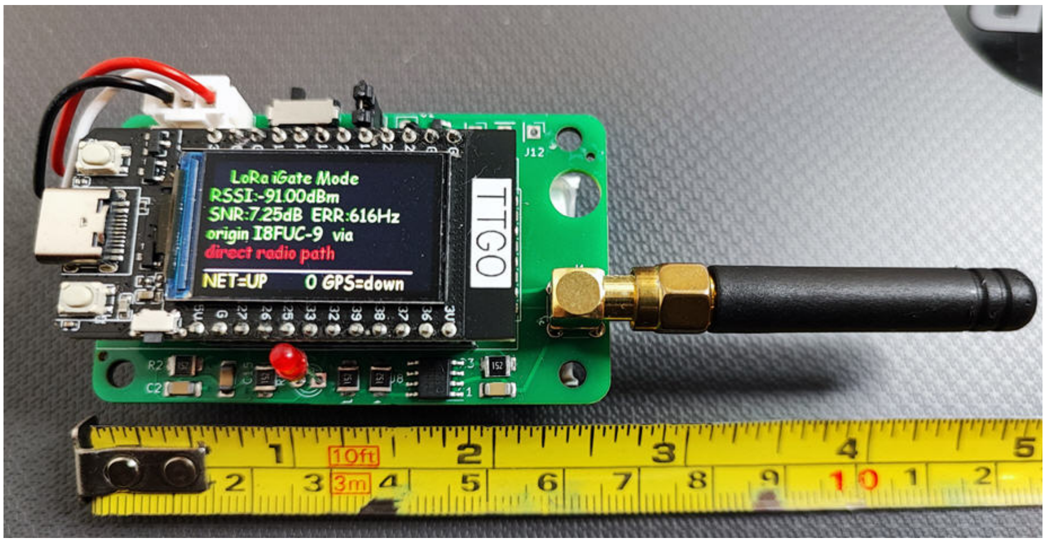
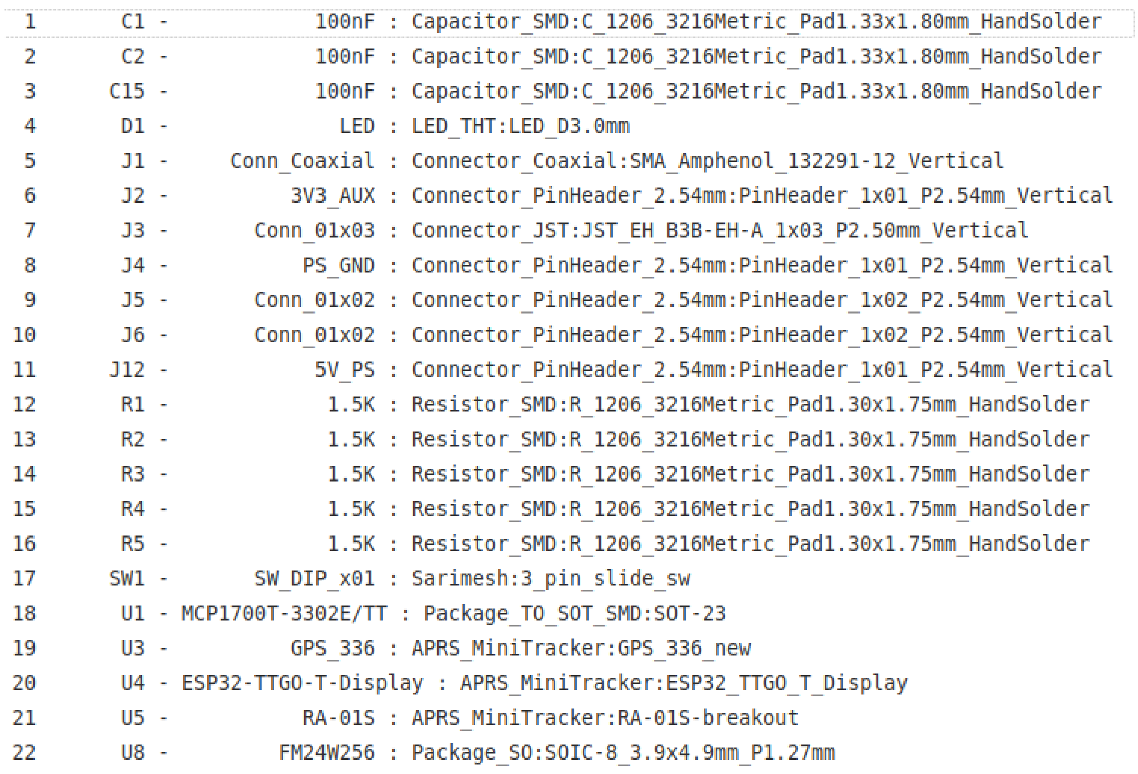


Figure 27: compact Tracker 2023 image

Following image 0 is the BOM of the device; please consider that we will make available a PCB already semi-assembled with all the SMD components; so only few components should be procured in order to complete the PCB assembly and no difficult SMD soldering in required (min. pin distance 2.0 mm).



```
 1      C1 -          100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.33x1.80mm_HandSolder
 2      C2 -          100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.33x1.80mm_HandSolder
 3     C15 -          100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.33x1.80mm_HandSolder
 4      D1 -            LED : LED_THT:LED_D3.0mm
 5      J1 -   Conn_Coaxial : Connector_Coaxial:SMA_Amphenol_132291-12_Vertical
 6      J2 -        3V3_AUX : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical
 7      J3 -      Conn_01x03 : Connector_JST:JST_EH_B3B-EH-A_1x03_P2.50mm_Vertical
 8      J4 -         PS_GND : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical
 9      J5 -      Conn_01x02 : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
10      J6 -      Conn_01x02 : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
11     J12 -          5V_PS : Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical
12      R1 -           1.5K : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
13      R2 -           1.5K : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
14      R3 -           1.5K : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
15      R4 -           1.5K : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
16      R5 -           1.5K : Resistor_SMD:R_1206_3216Metric_Pad1.30x1.75mm_HandSolder
17     SW1 -      SW_DIP_x01 : Sarimesh:3_pin_slide_sw
18      U1 - MCP1700T-3302E/TT : Package_TO_SOT_SMD:SOT-23
19      U3 -        GPS_336 : APRS_MiniTracker:GPS_336_new
20      U4 - ESP32-TTGO-T-Display : APRS_MiniTracker:ESP32_TTGO_T_Display
21      U5 -         RA-01S : APRS_MiniTracker:RA-01S-breakout
22      U8 -       FM24W256 : Package_SO:SOIC-8_3.9x4.9mm_P1.27mm
```

Figure 28: compact Tracker 2023 Bill of Material

6.8 LoRa_Dual_Mode_Tracker_2023_vr2 version (alias LoRa_DM)

This new PCB has been designed to be physically compatible with the year 2020 iGate-LoRa PCB; it is designed to use SMD components to make more space available for the new components required to implement the dual-radio architecture.

It is a modular device that is based on a standard ESP32 38 pin processor module and includes the specific modules required to implement a tracker/iGate device able to support the full set of Sarimesh SW Vr. 5.x functions; it includes support for 3 ambient sensors and a buzzer; it is arranged to support two displays (one OLED B/W 0.96" and a Color TFT 1.14").

The PCB has two main equipment variant:

- Color TFT display + GPS
- OLED BW Display + Ethernet interface

The two variants will need two different SW load images but will just differ for the different HW equipment supported.

The PCB can support two LoRa modules via suitable carrier boards; the LoRa modules must be mandatory second generation LoRa devices. Modules up to 2 watt output power can be supported. Typical equipment could be a 1 watt module for the primary radio and a 100 mwatt module for the secondary radio (only used as receiver).

The supported GPS is a last generation device compatible with all the GPS systems nowdays operational.

The PCB will expose two SMA antenna connectors to be connected either to two different antennas or to a single antenna via a power combiner/splitter.

The PCB is designed to fit in a standard plastic enclosure (https://secure.reichelt.com/it/it/kunststoffgehaeuse-serie-rm-100-x-130-x-50-mm-grau-rm2015m-p221333.html?&nbc=1) ; the enclosure can also contain the required power combiner just in case.

The design is completely jumper-less and there is no need for any change while using different LoRa module due to the LoRa carrier new design that performs suitable power supply adaptation.

The power supply is 12V via a standard power connector.

In the following 0 the schematics and the PCB 0 are presented as well as the device BOM 0 .

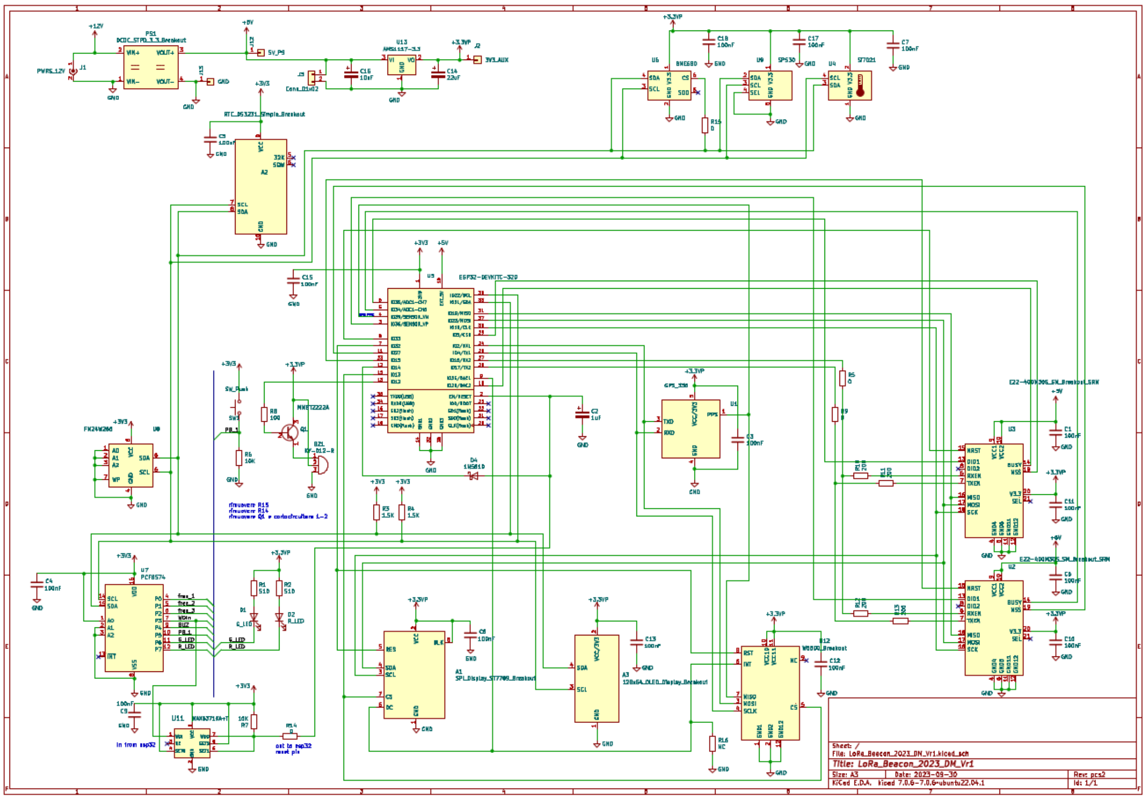


Figure 30: LoRa_Beacon_2023_DM Schematics

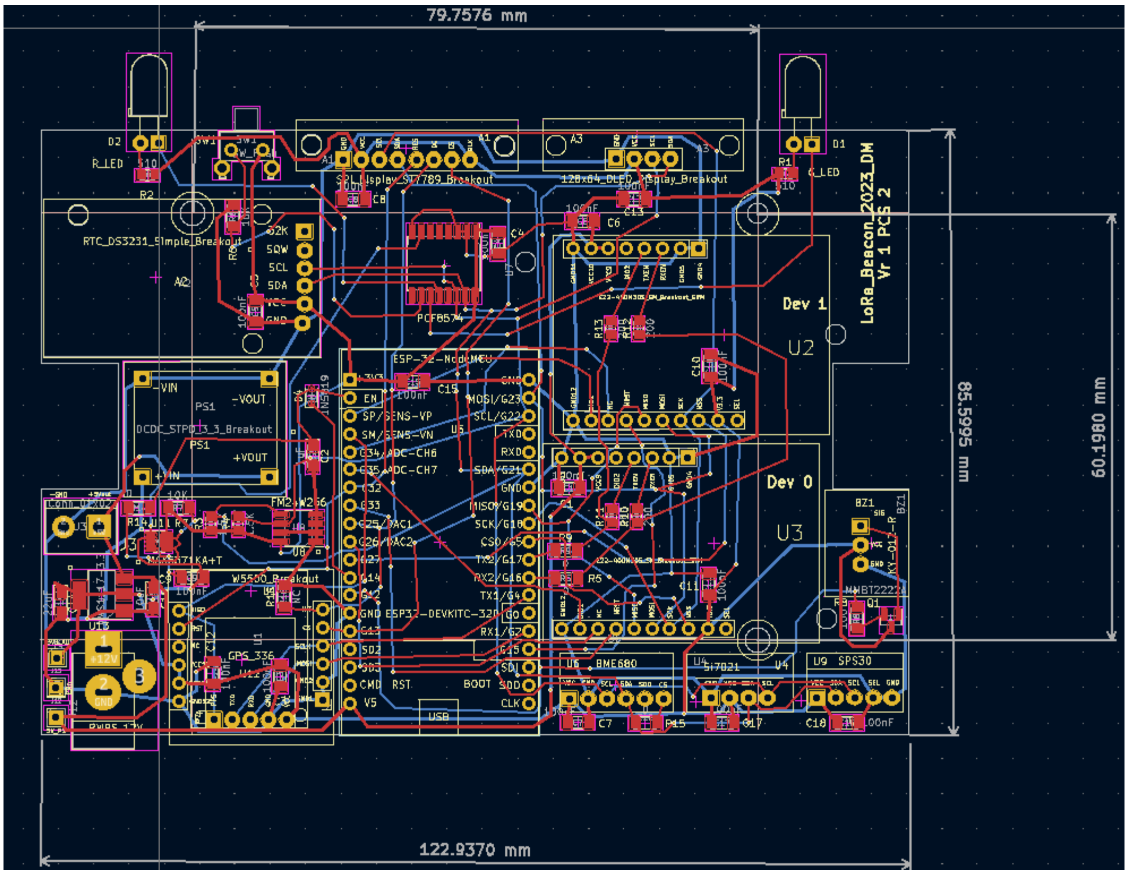
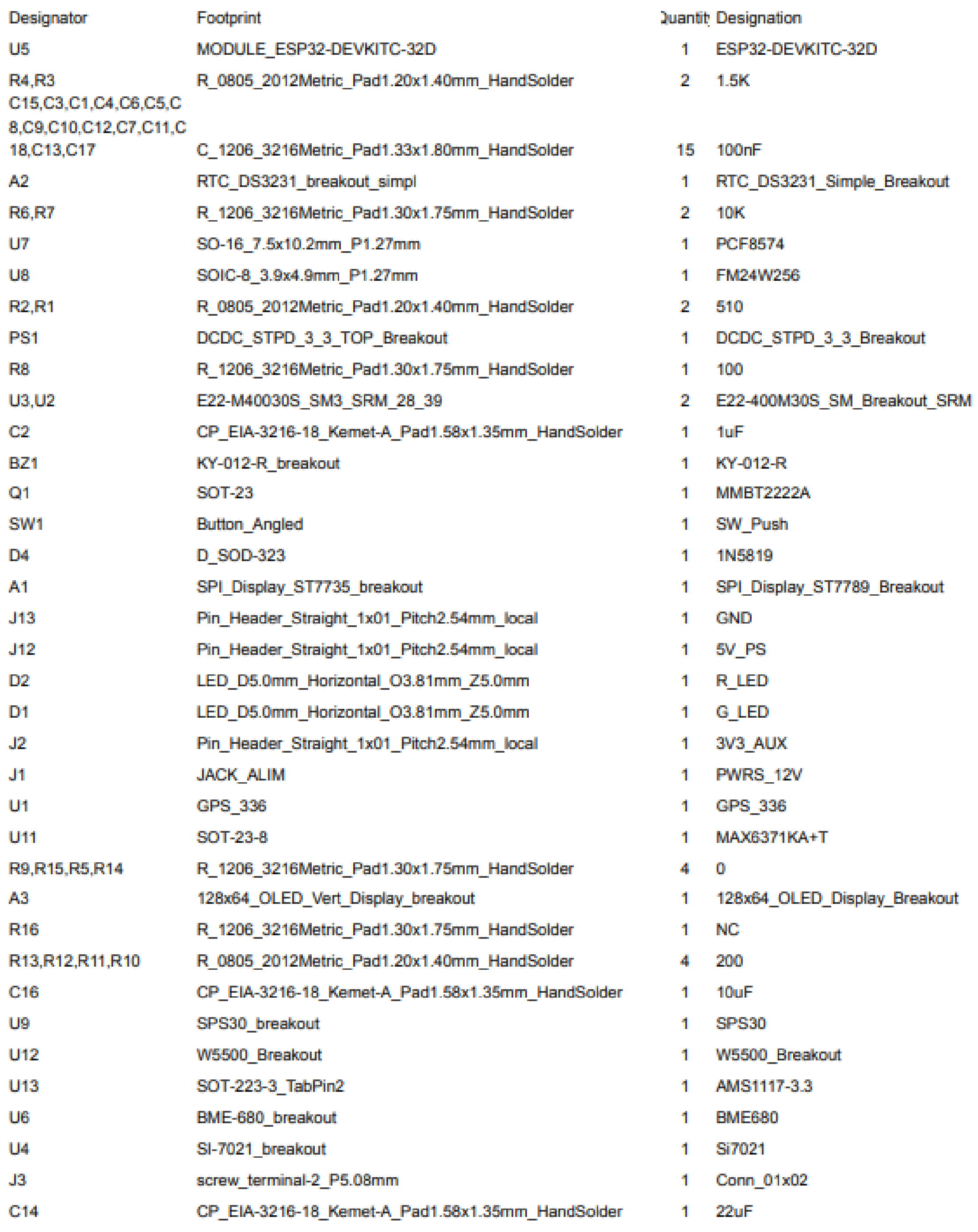


Figure 29: LoRa_Beacon_2023_DM PCB Layout



Designator	Footprint	Quantit	Designation
U5	MODULE_ESP32-DEVKITC-32D	1	ESP32-DEVKITC-32D
R4,R3	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	2	1.5K
C15,C3,C1,C4,C6,C5,C8,C9,C10,C12,C7,C11,C18,C13,C17	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	15	100nF
A2	RTC_DS3231_breakout_simpl	1	RTC_DS3231_Simple_Breakout
R6,R7	R_1206_3216Metric_Pad1.30x1.75mm_HandSolder	2	10K
U7	SO-16_7.5x10.2mm_P1.27mm	1	PCF8574
U8	SOIC-8_3.9x4.9mm_P1.27mm	1	FM24W256
R2,R1	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	2	510
PS1	DCDC_STPD_3_3_TOP_Breakout	1	DCDC_STPD_3_3_Breakout
R8	R_1206_3216Metric_Pad1.30x1.75mm_HandSolder	1	100
U3,U2	E22-M40030S_SM3_SRM_28_39	2	E22-400M30S_SM_Breakout_SRM
C2	CP_EIA-3216-18_Kemet-A_Pad1.58x1.35mm_HandSolder	1	1uF
BZ1	KY-012-R_breakout	1	KY-012-R
Q1	SOT-23	1	MMBT2222A
SW1	Button_Angled	1	SW_Push
D4	D_SOD-323	1	1N5819
A1	SPI_Display_ST7735_breakout	1	SPI_Display_ST7789_Breakout
J13	Pin_Header_Straight_1x01_Pitch2.54mm_local	1	GND
J12	Pin_Header_Straight_1x01_Pitch2.54mm_local	1	5V_PS
D2	LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm	1	R_LED
D1	LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm	1	G_LED
J2	Pin_Header_Straight_1x01_Pitch2.54mm_local	1	3V3_AUX
J1	JACK_ALIM	1	PWRS_12V
U1	GPS_336	1	GPS_336
U11	SOT-23-8	1	MAX6371KA+T
R9,R15,R5,R14	R_1206_3216Metric_Pad1.30x1.75mm_HandSolder	4	0
A3	128x64_OLED_Vert_Display_breakout	1	128x64_OLED_Display_Breakout
R16	R_1206_3216Metric_Pad1.30x1.75mm_HandSolder	1	NC
R13,R12,R11,R10	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	4	200
C16	CP_EIA-3216-18_Kemet-A_Pad1.58x1.35mm_HandSolder	1	10uF
U9	SPS30_breakout	1	SPS30
U12	W5500_Breakout	1	W5500_Breakout
U13	SOT-223-3_TabPin2	1	AMS1117-3.3
U6	BME-680_breakout	1	BME680
U4	SI-7021_breakout	1	Si7021
J3	screw_terminal-2_P5.08mm	1	Conn_01x02
C14	CP_EIA-3216-18_Kemet-A_Pad1.58x1.35mm_HandSolder	1	22uF

Figure 31: LoRa_Beacon_2023_DM Bill of Materials

Please consider that we will make available a PCB already semi-assembled with all the SMD components; so only few components should be procured in order to complete the PCB assembly and no difficult SMD soldering in required (min. pin distance 2.0 mm).

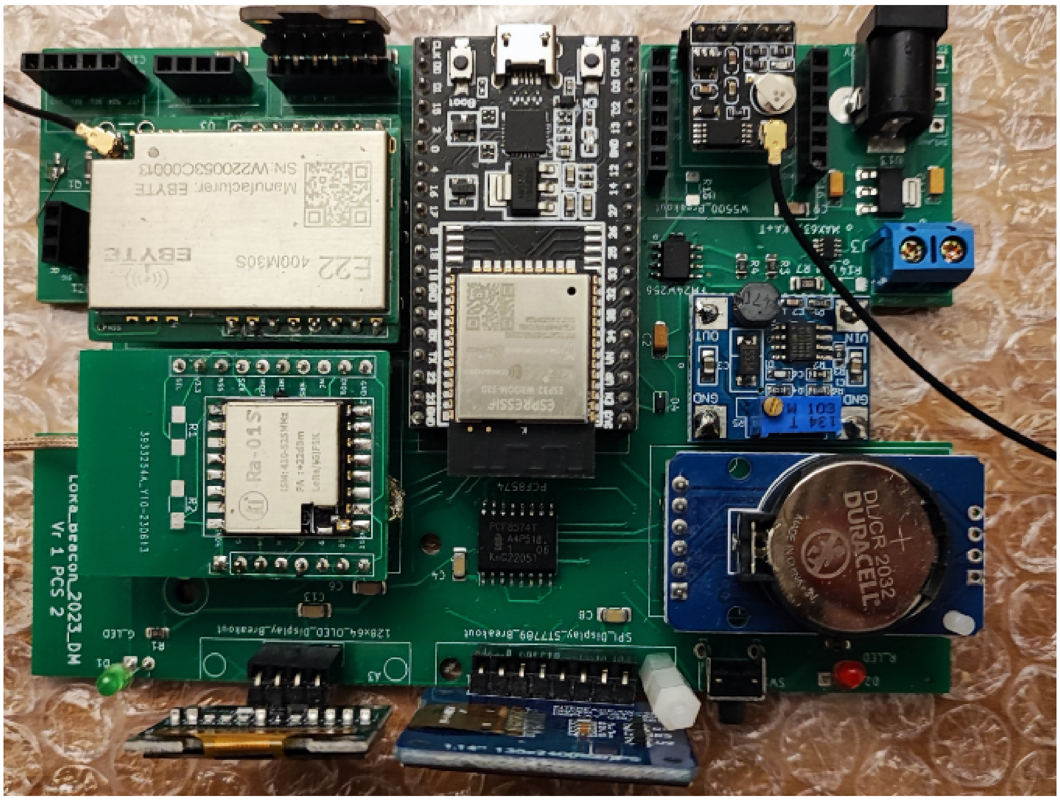


Figure 33: LoRa_Beacon_2023_DM GPS + Color Display

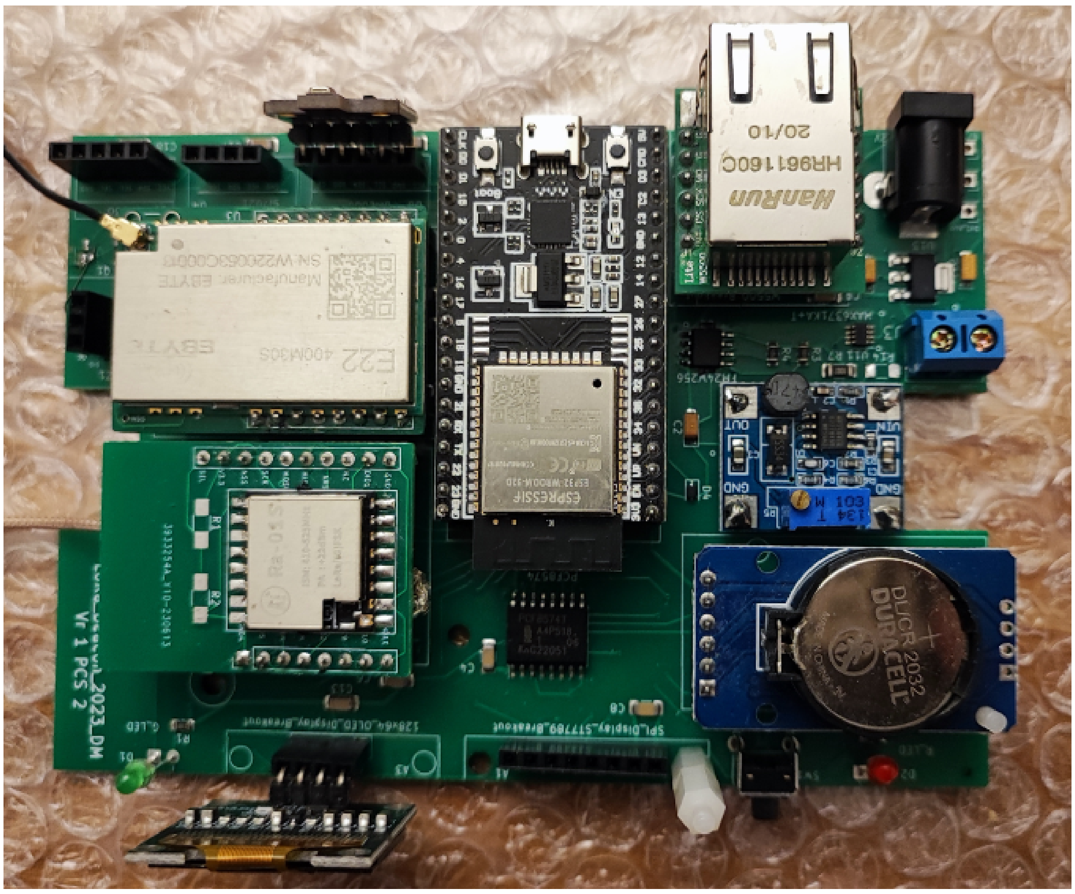
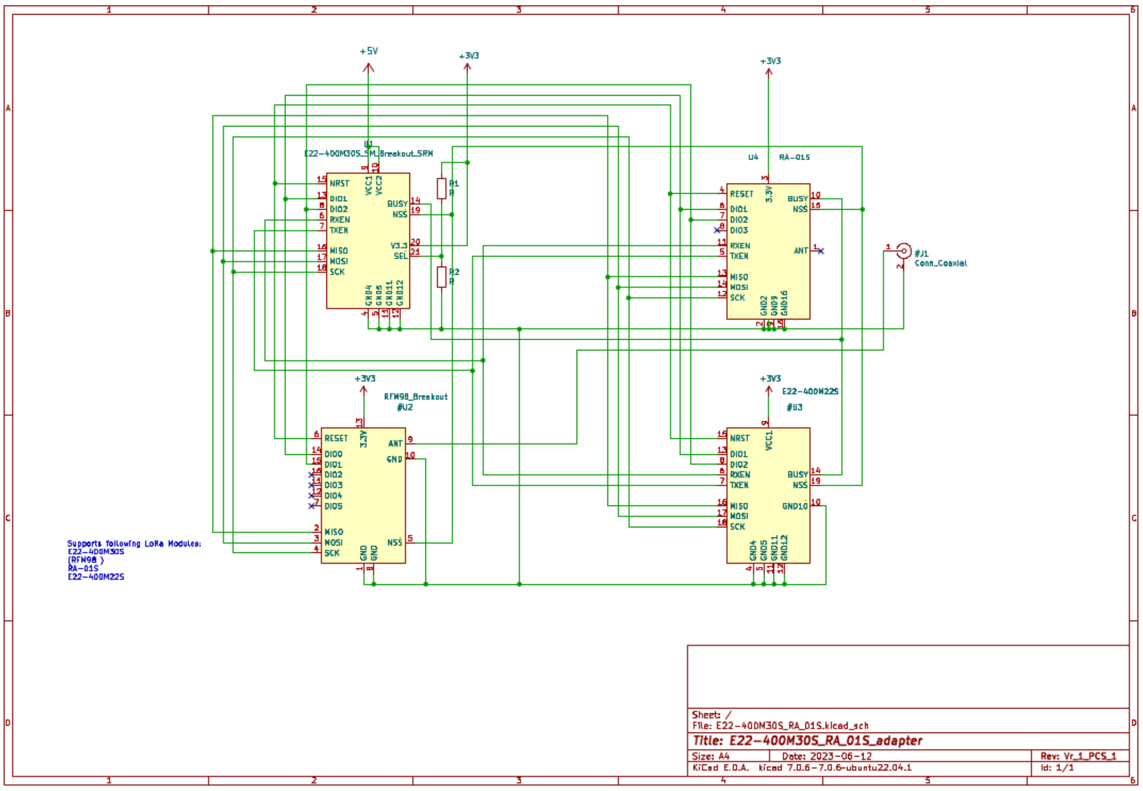


Figure 32: LoRa_Beacon_2023_DM Ethernet Interface + OLED Display

6.9 Carrier for LoRa radio modules year 2023 (LoRa carrier 2023)

A new carrier board gas been designed to allow to near-automatically adapts different LoRa modules to the main PCB. This imply a small pinout extension with respect to the year 2020 version and is backward compatible with 2020 LoRa carriers.

Following is additional infos regarding this new PCB:



+5V
+3V3
U1
E22-400M30S_SM_Breakout_SRM
U4
RA-01S
R1
R2
#J1
Conn_Coaxial
RFM98_Breakout
#U2
E22-400M22S
#U3
Supports following LoRa Modules:
E22-400M30S
(RFN98)
RA-01S
E22-400M22S
Sheet: /
File: E22-400M30S_RA_01S.kicad_sch
Title: E22-400M30S_RA_01S_adapter
Size: A4
Date: 2023-06-12
Rev: Vr_1_PCS_1
KiCad E.D.A. kicad 7.0.6-7.0.6-ubuntu22.04.1
Id: 1/1

Figure 34: LoRa_Carrier_2023 Schematics

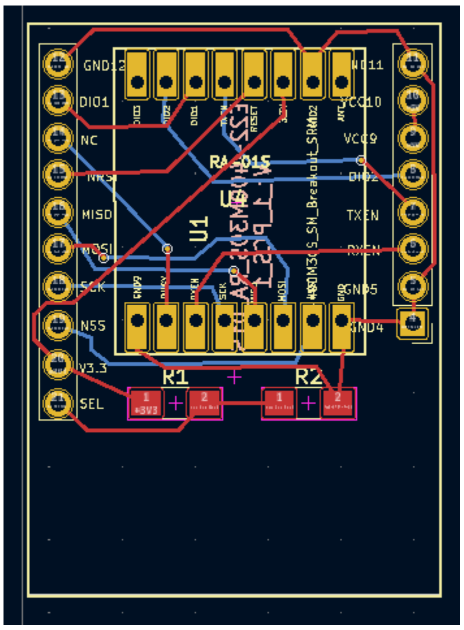


Figure 36: LoRa_Carrier_2023 PCB Lay-

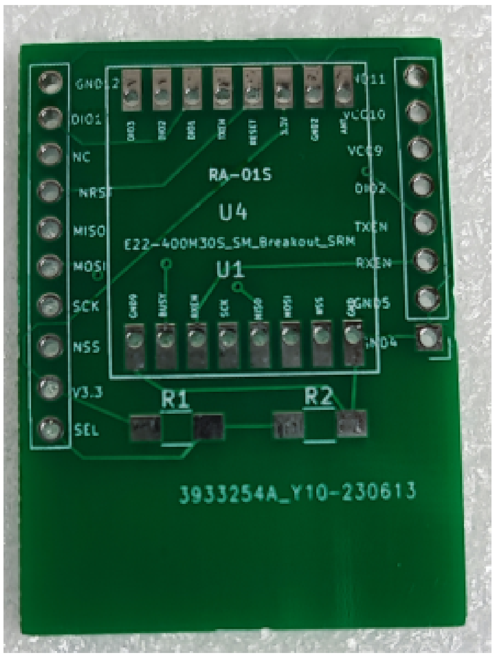


Figure 35: LoRa_Carrier_2023 PCB Layout

7 Initial SW installation

The microntroller used in the LoRa_Beacon project is the ESP32. It is a device with a very high integration scale uC with a significant amount of functions that are not described here for the sake of brevity; on the internet you can find a lot of documentation about it.

The processor can be purchased in the form of modules with standard 2.54mm pinout which contain, beyond the microcontroller, a series of other components including a flash memory which will be loaded with the SW program, a RAM memory to be used as working memory and a USB interface used for debug, direct monitoring and initial SW loading.

The following 0 shows the processor module and its physical interfaces. It is worth noting that there are several variants of this module on the market that differ in the number of pins and in the processor chip used; nearly all of our projects require to use the 38-pin module version with on-board (printed) WiFi antenna, as clearly shown in the following figures.

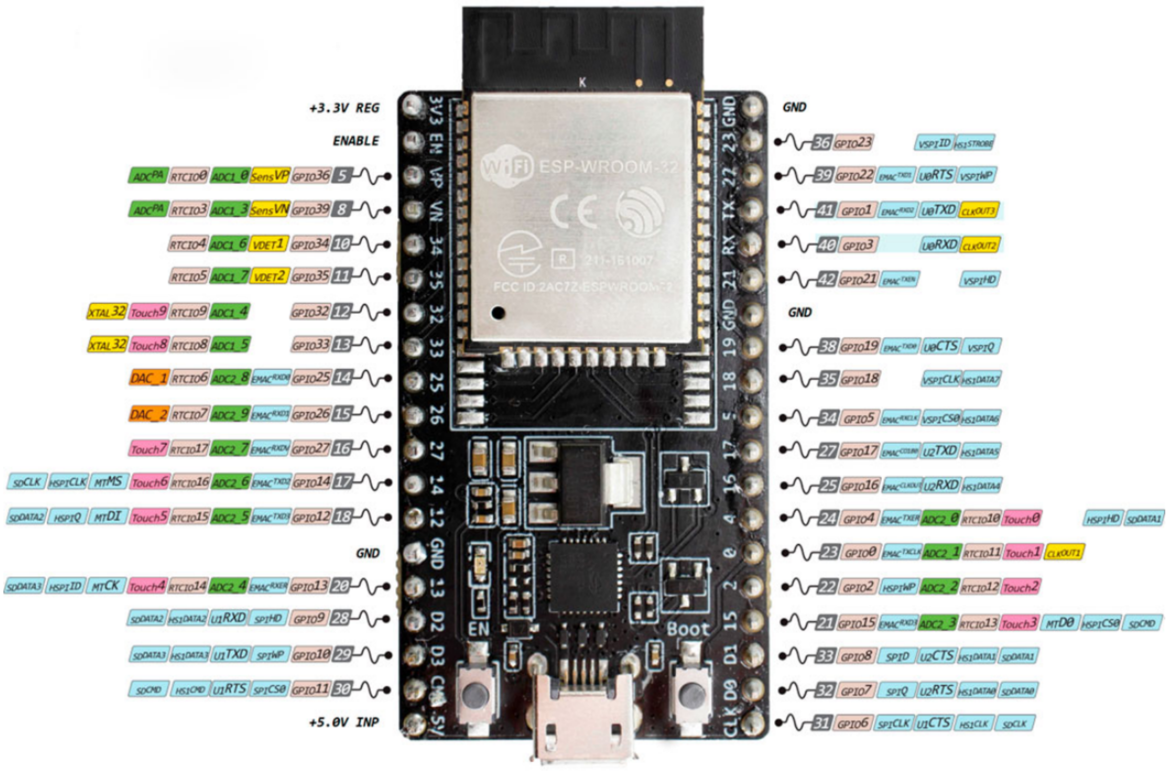


Figure 37: ESP32 processor module with related interfaces

At the time of purchase, the processor comes already equipped with a boot loader that allows the loading of the user SW via USB Interface and a PC equipped with a host GUI.

There are several possible development environments; the one used initially during the early stages of LoRa_Beacon project was based on the SW Arduino IDE platform; from the SW 4.x version the reference development environment has been migrated to the PlatformIO platform which is extremely more performing and user-friendly as setup and management; familiarization with this new platform involves a certain "learning curve" for users accustomed to Ardino which, however, will prove to be quite short and will significantly simplify the work for subsequent developments.

Only the first SW loading method is reported in this document: the big advantage of this method is that it allows you to load the SW onto the HW target **without necessarily having to set up any SW development environment** (eg. Arduino or PlatformIO) and without the need for any SW programming experience.

Obviously the second mode, based on the use of a PlatformIO development platform, is essential if you want to make changes to the SW or set particular options that cannot yet be managed directly via the device's graphic GUI interface.

7.1 First SW loading

The SW loading on the ESP32 processor requires a PC equipped with a Windows or Linux operating system and the proper host SW.

It is required to make the connection between the PC and the processor using a USB cable headed with appropriate connectors.

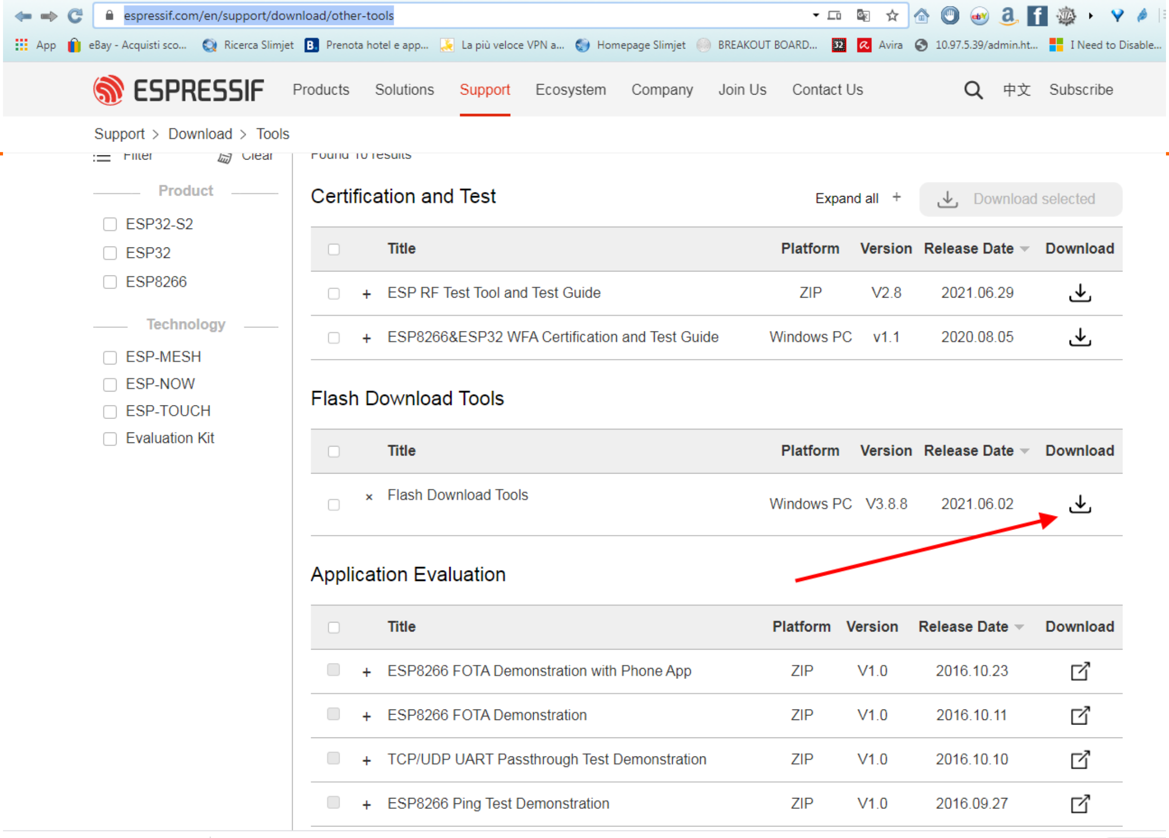
Upon connection via the USB cable, the processor will be powered via the cable itself and usually this operation will force Windows to automatically install the required serial port drivers.

To discover the identity of the serial port associated to the ESP32 module, simply explore the list of PC devices (via the control panel); on the Linux platform, recognition of the new interface is usually automatic.

In this phase the ESP32 module can be programmed even if it is not connected to the circuit on which it is to be used.

To download the programming tool from the internet, you can use the following URL: https://www.espressif.com/en/support/download/other-tools

The following Figure 38 shows the download page of the tool and indicates the version to download.


espressif.com/en/support/download/other-tools

Support > Download > Tools

Filter Clear

Product
- ESP32-S2
- ESP32
- ESP8266

Technology
- ESP-MESH
- ESP-NOW
- ESP-TOUCH
- Evaluation Kit

Found 10 results

Certification and Test Expand all + Download selected

	Title	Platform	Version	Release Date	Download
+	ESP RF Test Tool and Test Guide	ZIP	V2.8	2021.06.29	
+	ESP8266&ESP32 WFA Certification and Test Guide	Windows PC	v1.1	2020.08.05	

Flash Download Tools

	Title	Platform	Version	Release Date	Download
×	Flash Download Tools	Windows PC	V3.8.8	2021.06.02	

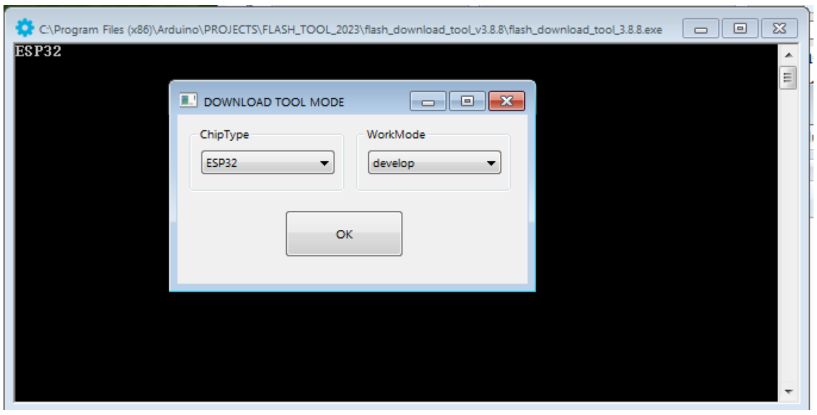
Application Evaluation

	Title	Platform	Version	Release Date	Download
+	ESP8266 FOTA Demonstration with Phone App	ZIP	V1.0	2016.10.23	
+	ESP8266 FOTA Demonstration	ZIP	V1.0	2016.10.11	
+	TCP/UDP UART Passthrough Test Demonstration	ZIP	V1.0	2016.10.10	
+	ESP8266 Ping Test Demonstration	ZIP	V1.0	2016.09.27	

Figure 38: Download page of the ESP32 programming tool

On the site http://iot-bits.com/esp32/esp32-flash-download-tool-tutorial/ it is also possible to find a small tutorial for using the tool; in this paragraph we will describe only the tool under a Windows environment. At the end a Linux installation tool will be also presented.

Once the Flash Download Tool has been downloaded on the PC, expand the relative archive and launch the “flash_download_tool_3.8.8.exe” file; then select from the small panel that appears "DOWNLOAD TOOL MODE" the values **"chip_Type = ESP32" and "WorkMode=develop"** ; a small panel will appear to be set as in the following figure:


C:\Program Files (x86)\Arduino\PROJECTS\FLASH_TOOL_2023\flash_download_tool_v3.8.8\flash_download_tool_3.8.8.exe

ESP32

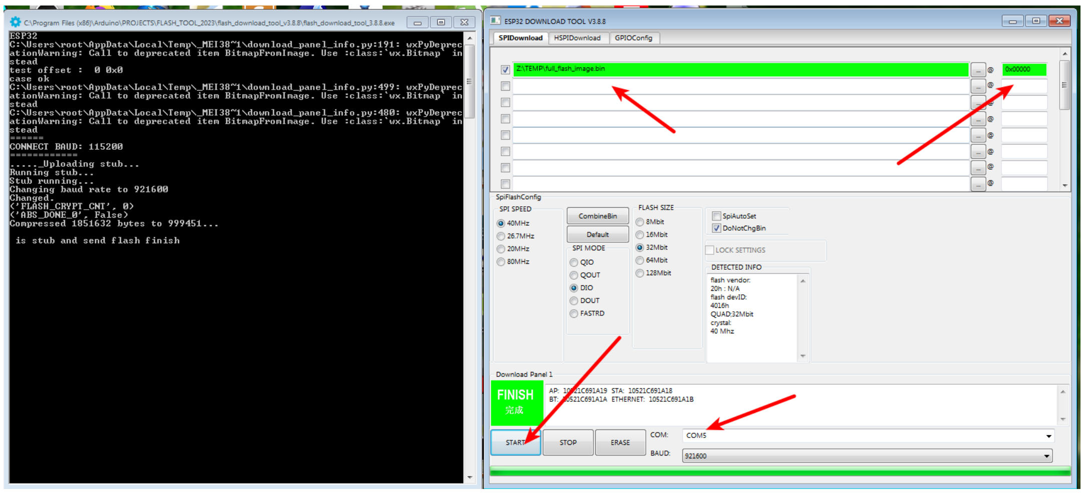
DOWNLOAD TOOL MODE

ChipType: ESP32

WorkMode: develop

Figure 39: Download Tool first screen

Going on a new screen will appear that requires few field to be selected or populated as in the next fig. 40



```
C:\Program Files (x86)\Arduino\PROJECTS\FLASH_TOOL_2023\flash_download_tool_v3.8.8\flash_download_tool_3.8.8.exe
ESP32
C:\Users\root\AppData\Local\Temp\_MEI38~1\download_panel_info.py:191: wxPyDeprec
ationWarning: Call to deprecated item BitmapFromImage. Use :class:`wx.Bitmap` in
stead
test offset :  0 0x0
case ok
C:\Users\root\AppData\Local\Temp\_MEI38~1\download_panel_info.py:499: wxPyDeprec
ationWarning: Call to deprecated item BitmapFromImage. Use :class:`wx.Bitmap` in
stead
C:\Users\root\AppData\Local\Temp\_MEI38~1\download_panel_info.py:480: wxPyDeprec
ationWarning: Call to deprecated item BitmapFromImage. Use :class:`wx.Bitmap` in
stead
======
CONNECT BAUD: 115200
============
......_Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
('FLASH_CRYPT_CNT', 0)
('ABS_DONE_0', False)
Compressed 1851632 bytes to 999451...

 is stub and send flash finish
```

ESP32 DOWNLOAD TOOL V3.8.8

SPIDownload | HSPIDownload | GPIOConfig

[✓] Z:\TEMP\full_flash_image.bin @ 0x00000

SpiFlashConfig

SPI SPEED: [✓ 40MHz] 26.7MHz 20MHz 80MHz

SPI MODE: QIO QOUT [✓ DIO] DOUT FASTRD

FLASH SIZE: 8Mbit 16Mbit [✓ 32Mbit] 64Mbit 128Mbit

SpiAutoSet [✓ DoNotChgBin]

LOCK SETTINGS

DETECTED INFO
flash vendor:
20h : N/A
flash devID:
4016h
QUAD;32Mbit
crystal:
40 Mhz

Download Panel 1

FINISH 完成

AP: 10521C691A19 STA: 10521C691A18
BT: 10521C691A1A ETHERNET: 10521C691A1B

COM: COM5
BAUD: 921600

Figure 40: SW Download tool on ESP32 processor setup

The serial port associated to the ESP32 interface (visible in Windows Control Panel) must be inserted in the COM window; for the other parameters, set them as in the figure. Set the BAUD value to 921600 ; if that doesn't work, try 115200.

The line in green is the one pointing to the first installation image to be loaded; it replaces the 5 modules that was required in SW ver. till 4.x to be loaded to various different address; with Vr. 5.x these 5 modules are already compacted in a single load module to be loaded always at address 0x000000 , thus simplyfing the setup process.

Just for information, the SW modules which compose the complete image are listed here below:

- Processor startup boot_app
- Boot_loader for loading the operating SW
- Application SW image
- Flash partition_table
- LittleFS partition

Make sure that no other related window is open at this stage, e.g. the IDE or other application that uses the same serial port.

By pressing the START key, the tool will load the indicated SW image onto the processor: in the command window of the tool, it is possible to follow the progress of the SW loading; if you do not notice the start of the green line on the lower edge of the main window, check the entered values and if necessary try to modify the BAUD value.

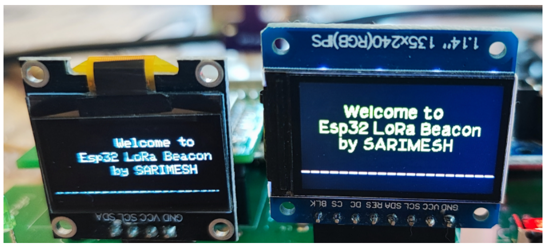
Once the SW image is uploaded mount the processor module on the main PCB and after few seconds check the display for signs of life :).

A number of welcome and advice screens will be displayed on the device display till a last screen that declares that the LoRa modules was successfully initialized; then a "Waiting for LoRa spots" screen will appear... this means that the device has been successfully setup and is running with the "Factory Defaults" configuration.

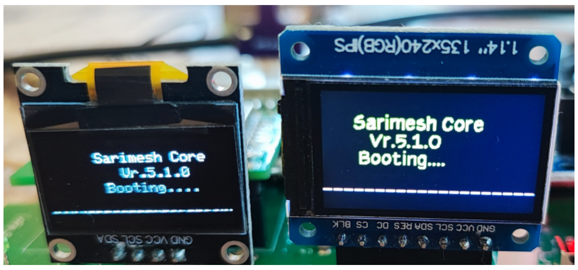
This configuration needs to be personalized with the actual user informations like will be displayed in the further chapters of this manual.

A detailed setup hands-on article is available at the following URL http://www.sar-imesh.net/2023/10/31/sperimentazione-lora-familiarizziamo-con-la-nuova-release-sw-5-x/ with reference to the installation over a TTGO T-Beam HW device; similar procedure is applicable also to the Sarimesh HW devices.

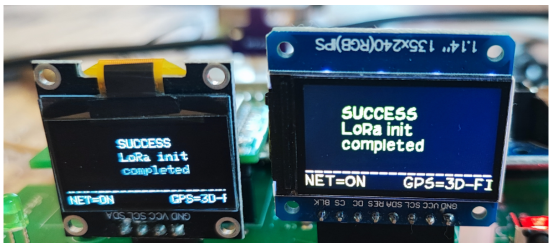
Following figures represent a typical first startup sequence of screens (fig. 41):



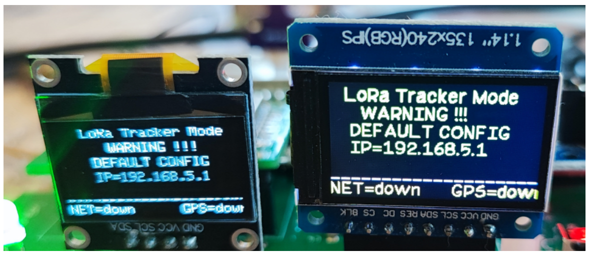
```
Welcome to
Esp32 LoRa Beacon
by SARIMESH
```



```
Sarimesh Core
Vr.5.1.0
Booting....
```



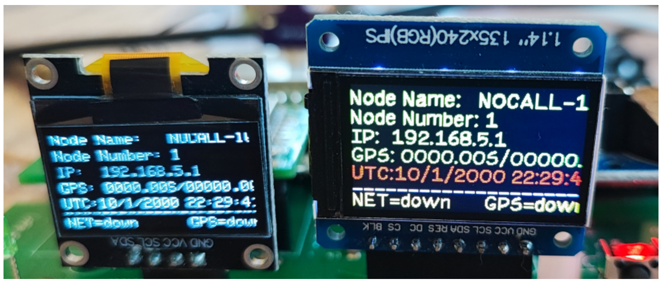
```
SUCCESS
LoRa init
completed
NET=ON     GPS=3D-FI
```



```
LoRa Tracker Mode
WARNING !!!
DEFAULT CONFIG
IP=192.168.5.1
NET=down   GPS=down
```

Figure 41: messages displayed during first boot after installation

Device default data can be seen by double short pushing the front panel button; following will appear (Fig. 42):



```
Node Name:  NOCALL-1
Node Number: 1
IP:  192.168.5.1
GPS: 0000.005/00000.
UTC:10/1/2000 22:29:4
NET=down    GPS=down
```

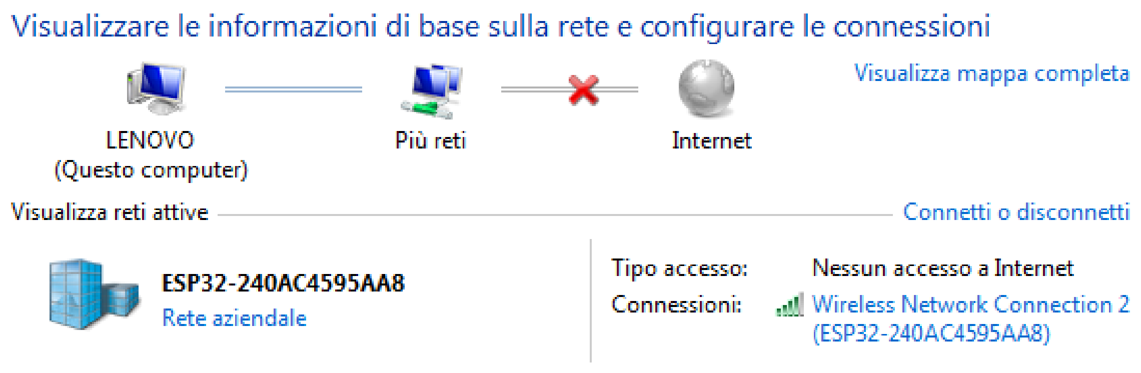
Figure 42: initial default device status display

After the default configuration is loaded, a new WiFi network called “ESP32-<mac address>” will appear. An example is reported in Fig. 43

Select this network on the PC (set DHCP enabled) and ESP32 will automatically assign an IP address to the PC **.**

It is also possible to access the device screen via a smart-phone: in this case by joining the device WiFi network an internet browser screen will popup on the smartphone screen automagically with the welcome screen already open.

If on PC, open an internet browser (chrome or firefox derivatives are warmly recommended...) and go the the device IP address that appears on the local device display.


Visualizzare le informazioni di base sulla rete e configurare le connessioni

Visualizza mappa completa

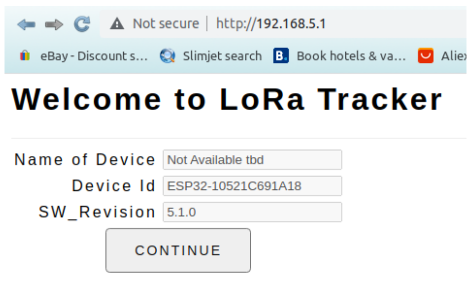
Visualizza reti attive — Connetti o disconnetti

ESP32-240AC4595AA8
Rete aziendale

Tipo accesso: Nessun accesso a Internet
Connessioni: Wireless Network Connection 2 (ESP32-240AC4595AA8)

Figure 43: example of Tracker SSID displayed in Windows WiFi management panel

By accessing to the Device GUI at http://192.168.5.1 following initial screen will appear (Fig. 44):


Not secure | http://192.168.5.1
eBay - Discount s... Slimjet search Book hotels & va... Aliex

Welcome to LoRa Tracker

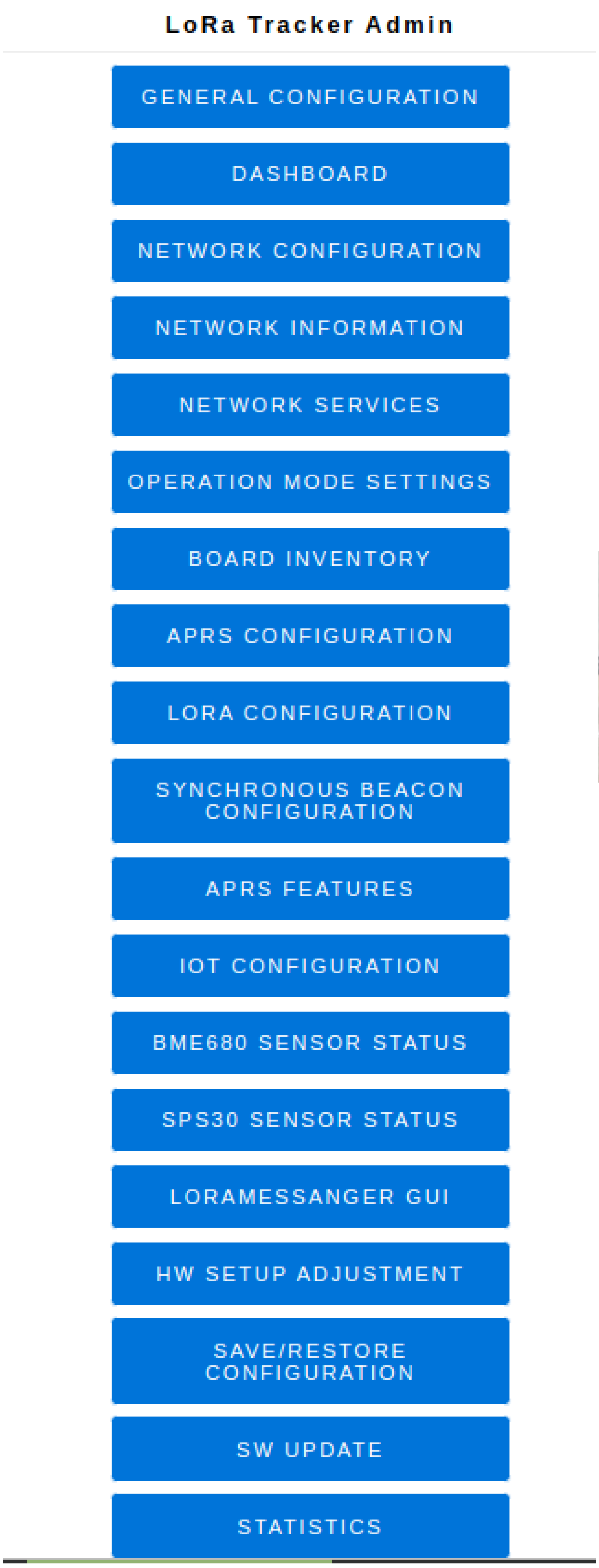
Name of Device	Not Available tbd
Device Id	ESP32-10521C691A18
SW_Revision	5.1.0

Figure 44: GUI welcome screen

By pushing the “Continue” button the main GUI screen will appear (Fig. 45):

With the SW 4.x version or 5.x, after the first loading on a brand new device, a default configuration is automatically loaded with a series of preset parameters in order to have a device already able to operate as a tracker. Obviously, for a concrete use it will be necessary to modify this configuration by replacing the default parameters with user parameters such as eg. the callsign of the device, the reference position (latitude/longitude), the Wi-Fi credentials (essential for the iGate type of operation) and the credentials of http://aprs.fi for forwarding the spots to APRS-IS. For customization, refer to the following sections of this document.

If the SW is loaded on a device previously used with the same or a different SW version, the old configuration of the device contained in the FRAM memory or in the flash for TTGO devices, of the processor will be maintained: this implies that if the new version of the SW is compatible with the previous one, the device will not need any adjustments to the configuration itself, while if the new version of SW is not completely compatible with the previous one, a punctual check and an adaptation may be necessary or a restoration of the default configuration could be necessary. Compatibility between versions will be highlighted in the release notes of the various SW versions that will follow.



LoRa Tracker Admin

GENERAL CONFIGURATION

DASHBOARD

NETWORK CONFIGURATION

NETWORK INFORMATION

NETWORK SERVICES

OPERATION MODE SETTINGS

BOARD INVENTORY

APRS CONFIGURATION

LORA CONFIGURATION

SYNCHRONOUS BEACON CONFIGURATION

APRS FEATURES

IOT CONFIGURATION

BME680 SENSOR STATUS

SPS30 SENSOR STATUS

LORAMESSANGER GUI

HW SETUP ADJUSTMENT

SAVE/RESTORE CONFIGURATION

SW UPDATE

STATISTICS

Figure 45: GUI main screen

With version 4.x and 5.x , dedicated compatibility checks are implemented and possibly indications are shown on the display to guide the setup.

Still in SW vr. 4.x and 5.x the restore mode to the default configuration has been simplified: now **when the device starts up and before any new message appears on the display there is a phase in which the two red and green LEDs will be lighted simultaneously for a few seconds... pressing the small button on the device in this phase restores the default conditions** and completely remove the old configuration of the device.

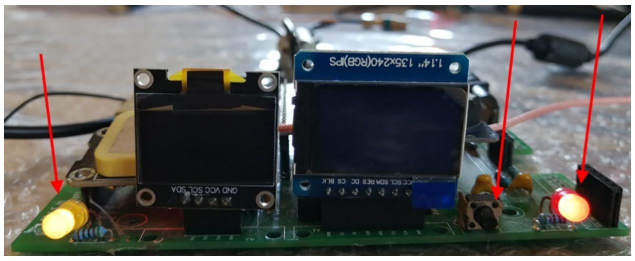
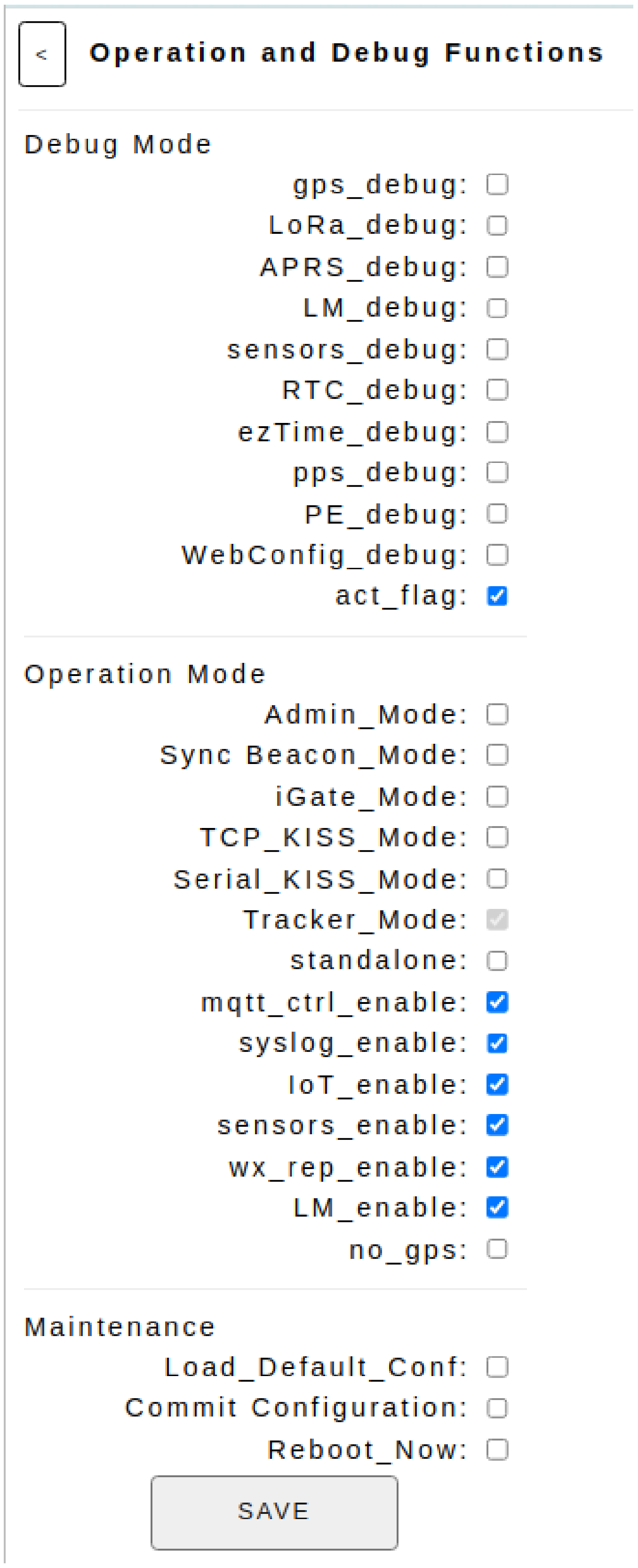


Figure 46 : Led During boot phase

For TTGO like devices missing the double red/green leds in order to restore factory defaults it is sufficient to switch off the device and then switch it on ... pushing the front panel button for 5-10 seconds will restore the factory default configuration blindly.

7.2 Initial setup of the LoRa_Beacon device (any version)

In its default configuration, the SW is not able to operate properly as it is necessary to insert some specific parameters, therefore a "configuration of the installed SW" phase is required .



< **Operation and Debug Functions**

Debug Mode

gps_debug:	☐
LoRa_debug:	☐
APRS_debug:	☐
LM_debug:	☐
sensors_debug:	☐
RTC_debug:	☐
ezTime_debug:	☐
pps_debug:	☐
PE_debug:	☐
WebConfig_debug:	☐
act_flag:	☑

Operation Mode

Admin_Mode:	☐
Sync Beacon_Mode:	☐
iGate_Mode:	☐
TCP_KISS_Mode:	☐
Serial_KISS_Mode:	☐
Tracker_Mode:	☑
standalone:	☐
mqtt_ctrl_enable:	☑
syslog_enable:	☑
IoT_enable:	☑
sensors_enable:	☑
wx_rep_enable:	☑
LM_enable:	☑
no_gps:	☐

Maintenance

Load_Default_Conf:	☐
Commit Configuration:	☐
Reboot_Now:	☐

SAVE

Figure 47: GUI "Operation Mode settings" page

To illustrate the configuration, first of all it's necessary to introduce the concept of "**operating modes** ".

The device can behave in different ways; these different operating modes are generally such that it is not possible to switch from one mode to another without rebooting the SW.

Some operations such as replacing the SW will require the freezing of some functions (e.g. the LoRa part and the GPS part). This is the so called "**Admin Mode**".

Clicking on **"OPERATION MODE SETTINGS"** button of the main page, the menu of fig. 47 will appear.

It is the **first section to be configured** after the initial loading of the SW.

A first part of the screen allows you to set a series of debug flags that can be used to track and fix operating anomalies or to set specific device equipment conditions. This section can initially be left in its default condition.

The second part of the screen allows you to set the operating mode for the device; various operating modes are possible which will be individually illustrated below.

The third part allows you to manually reboot the device without removing and reconnecting the power supply to the device, or also restore the default configuration for the device, or "commit" the present device configuration to make it "the permanent configuration". Something regarding the configuration strategy will be illustrated farther in this chapter.

The **"iGate_Mode"** mode prepares the device to operate as an APRS LoRa iGate and requires all the parameters present in the

"**APRS CONFIGURATION**" and "**APRS FEATURES**" screens to be checked and populated appropriately (see following...).

If this item is not selected, the "**Tracker APRS**" mode is automatically selected, which can also be customized using the same APRS Configuration screens as for the iGate Mode.

The "**BT_KISS_Mode**" , "**Serial_KISS_Mode**" ans "**TCP_KISS_Mode**" modes allow the device to be operated like a classic KISS TNC for interfacing with SW such as APRX or direwolf.

The "**Sync Beacon Mode**" is a special mode used for advanced LoRa experiments and will not be documented at present due to its too experimental status.

The "**Standalone**" item is intended to completely disable the upstream WiFi or Ethernet network connection; more on this further in this chapter.

The "**Syslog Enable**" item allows you to enable the automatic transfer of log messages to an external syslog server via the internet upstream connection if available; this function is intended for advanced device monitoring and requires a remote "**Syslog Server**" to be specified in the "**NETWORK SERVICES**" GUI page. This service uses the UDP port 514 for log messages transfer to the remote Syslog Server indicated. In standalone Operating mode or in case internet uptream connectivity is missing this service will not be operational.

The "**IoT_enable**" item is used to enable a series of IoT-type functions related to the sensors present on the device and is still to be documented. This service uses the MQTT protocol to interact with an external "**MQTT Broker**" in order to upload/download sensors data according to the available sensors on the device ; the use of this service requires to create a suitable "Application" on the remote MQTT Server in order to perform the wanted actions. This service is at present a proof of concept feature to be further documented and actively supported.

The "**mqtt_ctrl_enable**" item enables the remote control functions of the device via the mqtt protocol. This function is similar to the IoT functionality but is intended for remote transparent operation of the device via a suitable "**MQTT Broker**": the intended use is to be able to perform remote control actions on the device without requiring any setup on the internet gateway to wich the device is connected for internet access. At present it is still a proof of concept service and needs further documentation and development; anyway it is already operational to remotely reboot a device or acquire its status informations via the dashboard content upload on demand. This service requires an active internet upstream and is not operation if the internet connectivity for the device is missing.

The "**Sensors_enable**" item allow to enable the sensor support for the device. The actual sensors equipped on the device need first to be activated at the SW Build definition in order to be usable: this is required to limit the flash space occupied by the SW build for a specific device, if no sensor is supported. If Sensors support is present in the SW build in use, up to 3 different devices can be supported: they actually will be ready to provide their output data , but these data must be necessarily consumed by the SW; the actual use of these data can be of 3 types:

1. Sent via APRS to an external portal like aprs.fi, or just emitted at APRS network level to be displayed by any device able to do it
2. Sent via MQTT to an external MQTT Broker in order to feed any specified cloud application
3. to be displayed on a local GUI screen for local access or for access via the upstream internet connection using a browser interface.

These functions will be further documented in this document in a specific chapter.

The “**wx_rep_enable**” item will actually enable the sending of weather conditions data in the APRS beacons produced by the device; it requires that also the “**Sensors enable**” be activated. If no weather data is available for any reason this data is not sent over the APRS.

The “**LM_enable**” item allow to activate the new “**LoRa Messenger**” application; this application will be documented in a further specific chapter of this document.

The “**no_gps**” item will completely disable the GPS functionality and allow to operate for example a fixed device by simply setting a device position in the APRS GUI page; this allows to save GPS cost for a tipycal iGate fixed application.

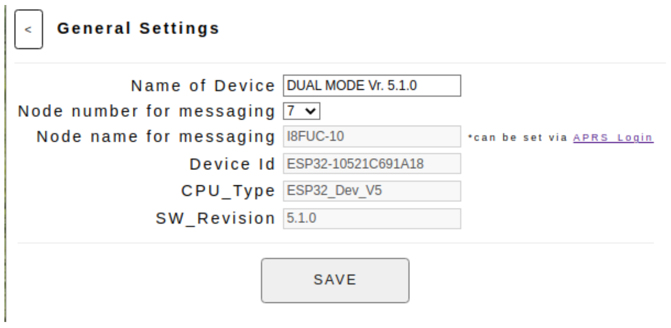
A third section of this GUI page is the “**Maintenance**” section; this section contains the following options:

1. **“Load_Default_Conf”** : this item allows to replace the present device configuration with the Default configuration in order to restore a clean device status just in case it gets corrupted or after a configuration session not producing the expected operation. The default configuration allows to setup a tracker mode operation with defaults for all the relevant data, so that a working device would be recovered in any case.
2. “**Commit configuration**”: this item allows to permanently freeze the present device configuration in the on board non volatile memory (either FRAM or Flash depending on the device type). More on this subject in a specific further chapter of this document.
3. “**Reboot_Now**”: just requires to reboot immediately the device via the GUI (possibly remote) operation.

All the selected items of this page will be saved in the “temporary device configuration” upon the “**SAVE**” button activation. After the save some of the actual selected features will be immediately activated, some other will not have any effect up to the next reboot (if the configuration is committed before actual reboot).

The “**immediate action**” features are actually the ones affecting the debug i.e. the ones present in the first section of this GUI page; the reason of this special operation way is that debug features generally are required just in case to check specific events or strange situations ... so they must not be subject to a restart of the device operation status to be used.

A further section to be configured initially is represented by the "GENERAL CONFIGURATION" item of the main GUI screen.



Name of Device	DUAL MODE Vr. 5.1.0	
Node number for messaging	7	
Node name for messaging	I8FUC-10	*can be set via APRS Login
Device Id	ESP32-10521C691A18	
CPU_Type	ESP32_Dev_V5	
SW_Revision	5.1.0	

Figure 48: GUI "General Configuration" page

This page contains some general data regarding the device in use and few items to be filled with users wanted data.

Any device will get automatically a “**Device_id**” derived from the WiFi device MAC Address that is a unique hexadecimal string fixed at ESP32 chip manufacturing. So this item makes possible to distinguish any device from all the others.

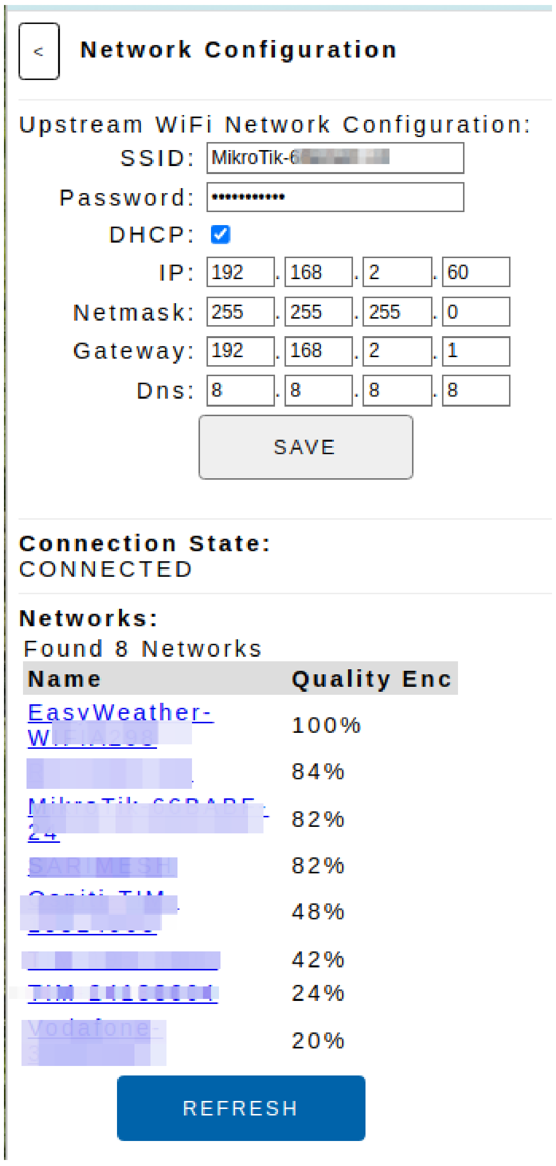
The “**Name of Device**” is a free field that the user can populate for identifying this device characteristics or specificities.

The “**Node number for messaging**” is a number in a specified range (default is 1-10) that need to be set for the LoRa Messenger application to be correctly operational. It must be unique in a group of devices that want to communicate each-other. Further infos on this in the specific chapter for LoRa Messenger application.

The “**Node Name for messaging**” is an alias of the previous item for mnemonic usage. It is actually always associated to the OM call sign as specified in the APRS Configuration for the device in use.

The “**CPU_Type**” reports the type of HW device .

The “**SW_Revision**” is the actual SW release number installed on the device.


< **Network Configuration**

Upstream WiFi Network Configuration:

SSID: MikroTik-6[redacted]

Password: ••••••••••••

DHCP: ☑

IP: 192 . 168 . 2 . 60

Netmask: 255 . 255 . 255 . 0

Gateway: 192 . 168 . 2 . 1

Dns: 8 . 8 . 8 . 8

SAVE

Connection State:
CONNECTED

Networks:
Found 8 Networks

Name	Quality	Enc
EasyWeather-[redacted]	100%	
[redacted]	84%	
[redacted]	82%	
[redacted]	82%	
[redacted]	48%	
[redacted]	42%	
[redacted]	24%	
[redacted]	20%	

REFRESH

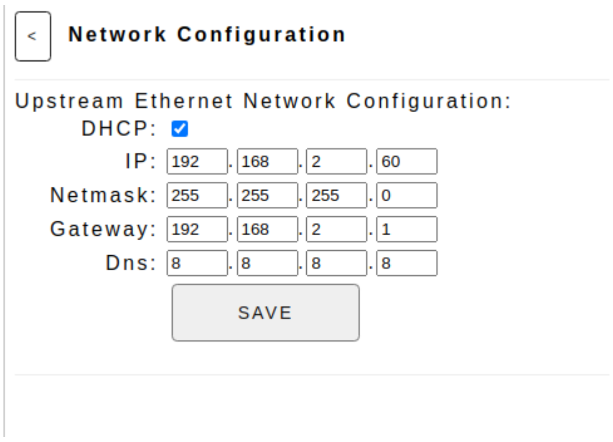
Figure 49 : GUI "Network Configuration" page

A further section to be set initially is the "**NETWORK CONFIGURATION”.** This section allows to specify the network addressing for the device; depending on the HW configuration the device can have a WiFi interface or an Ethernet interface available for interacting with the rest of the world; if the Ethernet interface is in place, the WiFi interface is explicitely disabled.

In both cases the device can operate in two different modalities regarding the “**upstream connection**” toward internet: “**Standalone**” or “**Not Standalone**”; if in standalone operation the device will assume a local address (respectively 192.168.5.1 or 192.168.4.1) and will operate as a device completely isolated from the internet.
This is a typical situation for a mobile device while not close to an external accesspoint. Mainly for the WiFi case this mode of operation allows to reduce the power consumption of the device thus extending the battery life while isolated from the external power source.

When in **Standalone** operation mode the device can be controlled by directly connecting a PC or a similar device to the WiFi HotSpot presented by the device (if wifi interface is in place) or via a LAN connection if the ethernet interface is in place. For the Ethernet interface by design the device will not operate any DHCP server for security reasons, so the controlling PC must be set manually to an IP address in the network IP 192.168.4.xx/24 (i.e 192.168.4.123 with netmask 255.255.255.0). For WiFi, thanks to the private SSID associated with any device a DHCP Server can be operated on the device , thus the PC or smartphone addressing will be automatically acquired from the WiFi AP DHCP server.

When using a smarthphone (IOS or Android) for connecting to the device AP, as soon as the smartphone gets connected to the device network, automagically a browser windows will popup presenting the device welcome GUI screen.



Upstream Ethernet Network Configuration:

DHCP:	✓			
IP:	192	168	2	60
Netmask:	255	255	255	0
Gateway:	192	168	2	1
Dns:	8	8	8	8

Figure 50 : GUI "Network Configuration" page

When in Non Standalone operation for both interface types the device can be set to a static address if wanted or to DHCP mode in order to get an IP addressing automatically from an external DHCP server available on the external network.

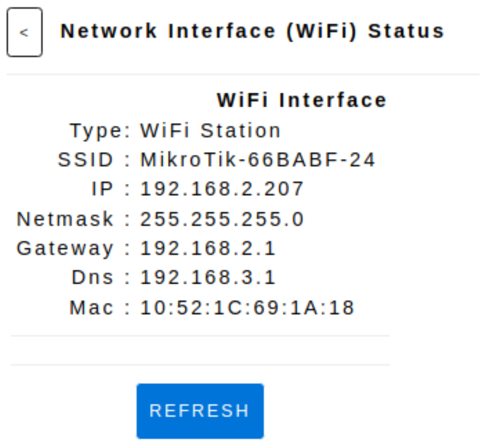
Fig. 49-50 document the GUI page to set the IP addressing for the device in both the interface type cases.

For the WiFi case the GUI screen will present in the bottom part a list of discovered 2.4Ghz networks available for connection, with an indication of the signal level; by selecting any entry, the related field in the upper part of the screen will be filled ; please remember to set a valid password if required. Then the "SAVE" button will save the selected data in the temporary device configuration.

The list of available networks could be empty... just wait a little bit to get it populated...

The "REFRESH" key is used to reacquire the list of available WiFi networks.

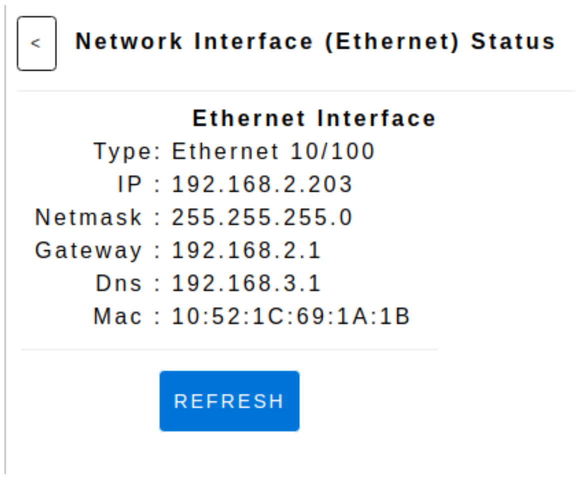
To find out the status of the WiFi or ethernet network connection, there is a "**NETWORK INFORMATION**" pageI: this page will provide the values currently in use by the device for its connection to the local network and to the internet.



WiFi Interface

```
   Type: WiFi Station
   SSID : MikroTik-66BABF-24
     IP : 192.168.2.207
Netmask : 255.255.255.0
Gateway : 192.168.2.1
    Dns : 192.168.3.1
    Mac : 10:52:1C:69:1A:18
```

Figure 52: GUI "Network Information"



Ethernet Interface

```
   Type: Ethernet 10/100
     IP : 192.168.2.203
Netmask : 255.255.255.0
Gateway : 192.168.2.1
    Dns : 192.168.3.1
    Mac : 10:52:1C:69:1A:1B
```

Figure 51 : GUI "Network Information" page

To setup the Standalone or Not Standalone operation mode a mechanism is required in order to change this attribute also without the possibility to manage the device via a PC o similar tool due to the fact that device could be in a status not reachable via network connection.

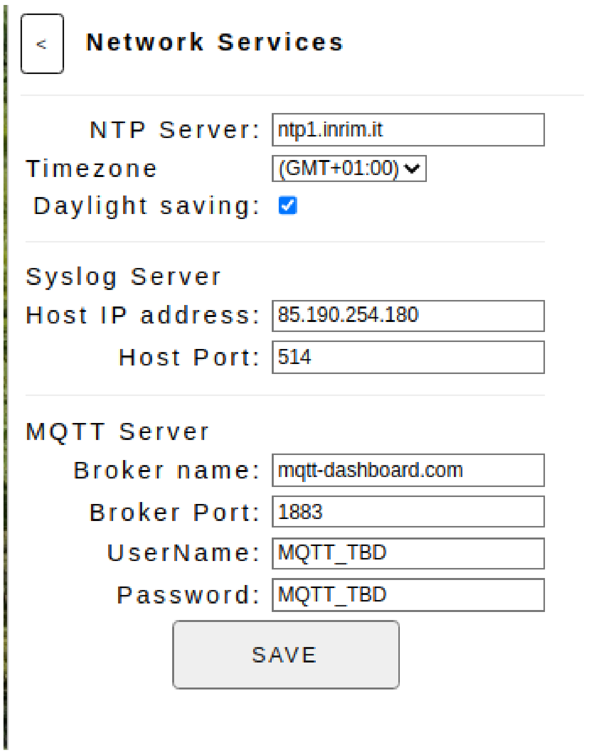
The implemented mechanism make use of the small front panel button: by keeping pressed the button a number of screens will appear timed on the display presenting a set of possible action: by releasing the button the presently displayed action will be executed.

This mechanism is made available to activate or disactivate the Standalone mode without any additional terminal required. Please consider that in order to make the new mode active a reboot could be required (will be indicated on the screen just in case).

The device reboot function can be performed either by switching the device off and on again, or by keeping the key pressed until the wording "**Reboot Now**" appears on the display.

To complete the configuration of the device's network parameters, it is necessary to specify a "**Network Time Server**" which eventually be used for synchronizing the local time of the device which will be used for the timestamping of the logs and messages.

For this purpose, there is a "**NETWORK SERVICES**" page accessible from the main page shown in 0 ; ths page allows to set a number of services that will use the upstream internet connection if available for various functions.



NTP Server: ntp1.inrim.it
Timezone (GMT+01:00)
Daylight saving: ✓

Syslog Server
Host IP address: 85.190.254.180
Host Port: 514

MQTT Server
Broker name: mqtt-dashboard.com
Broker Port: 1883
UserName: MQTT_TBD
Password: MQTT_TBD

Figure 53: GUI "Network Services" page

The first function is to set an "**NTP Server**" for device time synchronization via the NTP protocol: this service allows , if the internet upstream is available, to set the local time with a good accuracy for any time sensitive service; actually it will be used for stamping permanent local log records keept in the FRAM (if available on the HW), and for stamping eventually the LoRa or internet traffic produced by the device.

Other services could need an accurate time setting: in particular the "Synchronous Beacon" Service will need such time accuracy.

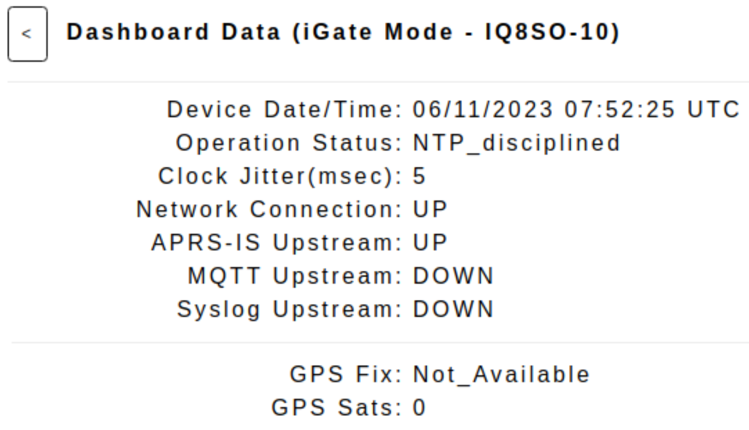
The setting requires to enter a NTP Server ad a timezone indication.

In any case if also an active GPS is available , the latter will take precedence as source of time synchronization.

In addition the NTP (or GPS) will be used to synchronize the local RTC (**RealTimeClock**) chip (if available on the HW) in order to be able to have a nearly correct "**time of day**" also if the an exernal synch source is not available at all; this is for example the case for a remote fixed device without GPS situated in a place where internet connectivity is not available.

Another network service is the "**Syslog Server**" connection: this service allows to send, if the uppstream internet connection is available, log messages generated by the device to a remote Syslog Server to collect informations for monitoring the installed base of devices or for any other statistics. The service will use UDP messages on port 514 as a default.

Another service available is the "**MQTT Server**" connection: the MQTT protocol is very usefull for IoT like applications and can also be used for remote operation of devices without requiring any setup action on the remote internet gateway to wich the monitored device is attached. In our case the service allows to perform transparently a number of remote actions , if the target device is upstream connected to internet, like getting Dashboard status, remote rebooting a device etc.



Device Date/Time: 06/11/2023 07:52:25 UTC
Operation Status: NTP_disciplined
Clock Jitter(msec): 5
Network Connection: UP
APRS-IS Upstream: UP
MQTT Upstream: DOWN
Syslog Upstream: DOWN

GPS Fix: Not_Available
GPS Sats: 0

Figure 54 : GUI "Dashboard" page (part 1)

To find out the internet connection status of the device and its main operating characteristics, it is possible to use the "DASHBOARD" screen on the main page of the GUI.

This screen, which will be extensively described in detail next in this chapter, will show also a number of informations and statistics regarding the actual operational status of the main basic services and the time synchronization status for the device.

The first section of this “Dashboard” will report the actual subsystems status for the device; the “**Device (local) Date/Time**” can be derived from 3 different synch sources:

1. **GPS** time if available via a good GPS Fix (highest precision)
2. **NTP** via upstream internet connection (intermedie precision)
3. local **RTC** in case no better reference is available.

In order to have the RTC synch operational it is essential that there is at least a first pass trought one of the other synch methods in order to allow the local RTC chip to be automagically setup with a real time source at least once. Then the RTC time/date is keept in a dedicated RTC-backup-battery, so that it survives to powering status of the device.
The RTC synch source allows, in addition, to have the device Date/time always available immediately also after the device power-on without any other tyme sync source available or manual action required.

The “**Clock Jitter”** is computed in realtime and provides an evaluation of the present clock stability and precision.
The other items in the screen are self-explained by their name.

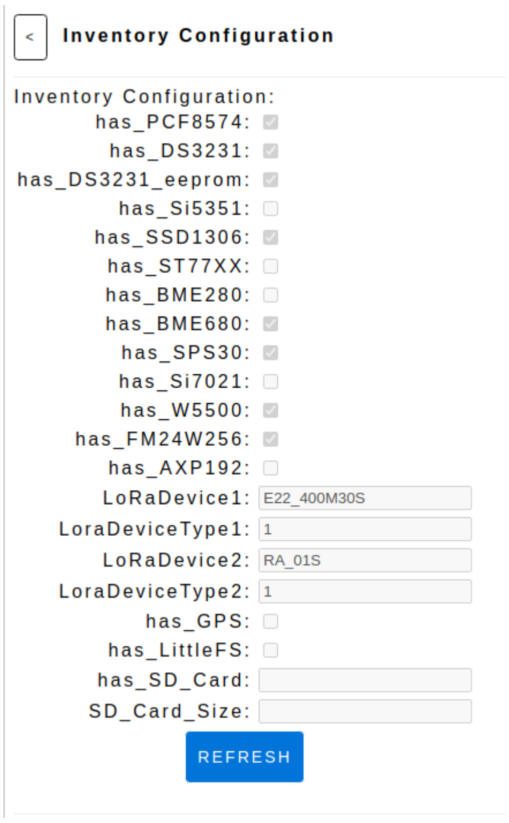
A very important section to check after a SW installation is the “ **BOARD INVENTORY** ” page accessible from the GUI main page.

Due to the modular type of the HW and SW of the Sarimesh Core SW, any device can have a variable configuration in terms of a set of HW “modules” and a set of “SW modules”: so it is necessary to be able to know in realtime what the present in use device is capable to do.

The inventory page of the GUI provides a synthesis of what is actually available and usable on a particular PCB sample due to its HW/SW configuration.
Most of the optional plugin-modules will be automatically recognized during the start-up phase; other must be declared in the Build Definition File used for SW image construction. The result of this phase is summarized in the screen above.

The inventory data are collected based on the actual SW build definition and the actual HW derived data... so in order to see as “available” any module it must be physically present in the HW (if it is an HW item) and be enabled in the actual SW Build definition. This latter condition is required in order to be able to create SW builds with minimum memory size just in case any HW/SW module does not need to be supported; if a module is disabled in the Build Definition some pieces of SW are NOT inserted in the SW image and this reduces the image size and flash occupancy.

In addition this page allows in the installation phase to have an explicit evidence of the HW pieces detected so that it is easy to find possible installation mistakes.



Inventory Configuration:

has_PCF8574:	✓
has_DS3231:	✓
has_DS3231_eeprom:	✓
has_Si5351:	☐
has_SSD1306:	✓
has_ST77XX:	☐
has_BME280:	☐
has_BME680:	✓
has_SPS30:	✓
has_Si7021:	☐
has_W5500:	✓
has_FM24W256:	✓
has_AXP192:	☐
LoRaDevice1:	E22_400M30S
LoraDeviceType1:	1
LoRaDevice2:	RA_01S
LoraDeviceType2:	1
has_GPS:	☐
has_LittleFS:	☐
has_SD_Card:	
SD_Card_Size:	

Figure 55 : GUI "Board Inventory" page

A very important feature is the one indicated as "**has_FM24W256**": this is the FRAM memory dedicated to the device configuration data; the absence of recognition of this device prevents the saving of the configuration data in FRAM for Sarimesh original PCB designs. For TTGO like devices this feature in not available and thus the configuration data needs to be stored in the on-board flash LittleFS.

Most of the items in this screen are self - explained by their name; for LoRa devices the LoRaDeviceType is 0 for first generation LoRa modules and 1 for second generation LoRa modules.

The ethernet interface is associated to the W5500 module so in order to have an equipped ethernet internet the has_W5500 flag must be checked.

A particular mention to the "act_flag" flag, or activity flag: it is a flag to always be left activated.

At this point we can say that our new device is nearly populated with most of the pre-conditions to be able to perform the real work!!!!!

We have only to set now the relevant data for the top level services we would like to use; there are presently the following presentation services available:

1. APRS service
2. LoRa Messenger service
3. IoT services
4. Synchronous Beacon Service

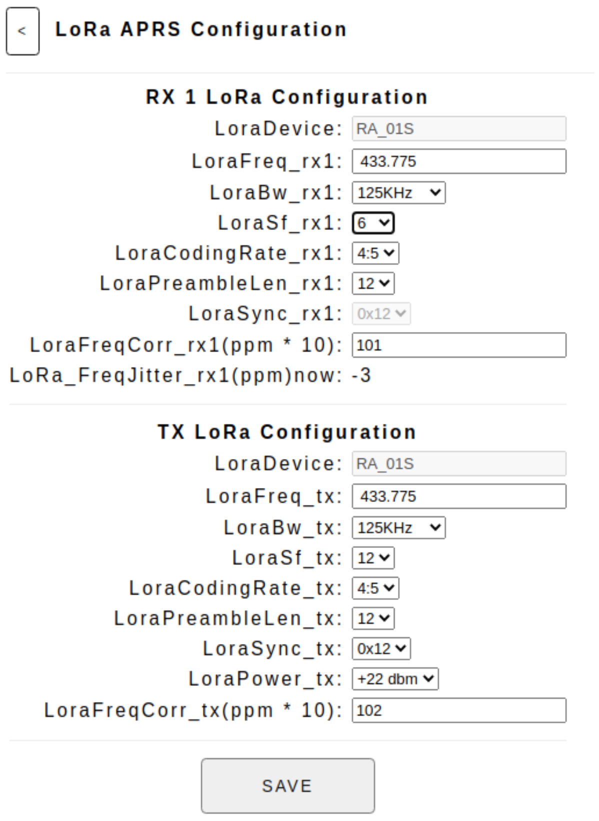
The 3) and 4) services are still under development and will not be described herein for now; the first two will be extensively described in the following chapters in this document.

Some of the top level services will require LoRa subsystem to be up&running, while others (like IoT or sensor related services) will not require LoRa availability.

7.3 LoRa subsystem setup (any version)

The LoRa subsystem is a mandatory service to be setup in order to have operational the services depending on it.

The Sarimesh SW is designed to have the LoRa operation as a subsystem practically independent from the top level services in order to allow its usage by different services. So it needs to be setup by specifying the actual LoRa related parameters that are available to the top level services to be used.



RX 1 LoRa Configuration

LoraDevice: RA_01S
LoraFreq_rx1: 433.775
LoraBw_rx1: 125KHz
LoraSf_rx1: 6
LoraCodingRate_rx1: 4:5
LoraPreambleLen_rx1: 12
LoraSync_rx1: 0x12
LoraFreqCorr_rx1(ppm * 10): 101
LoRa_FreqJitter_rx1(ppm)now: -3

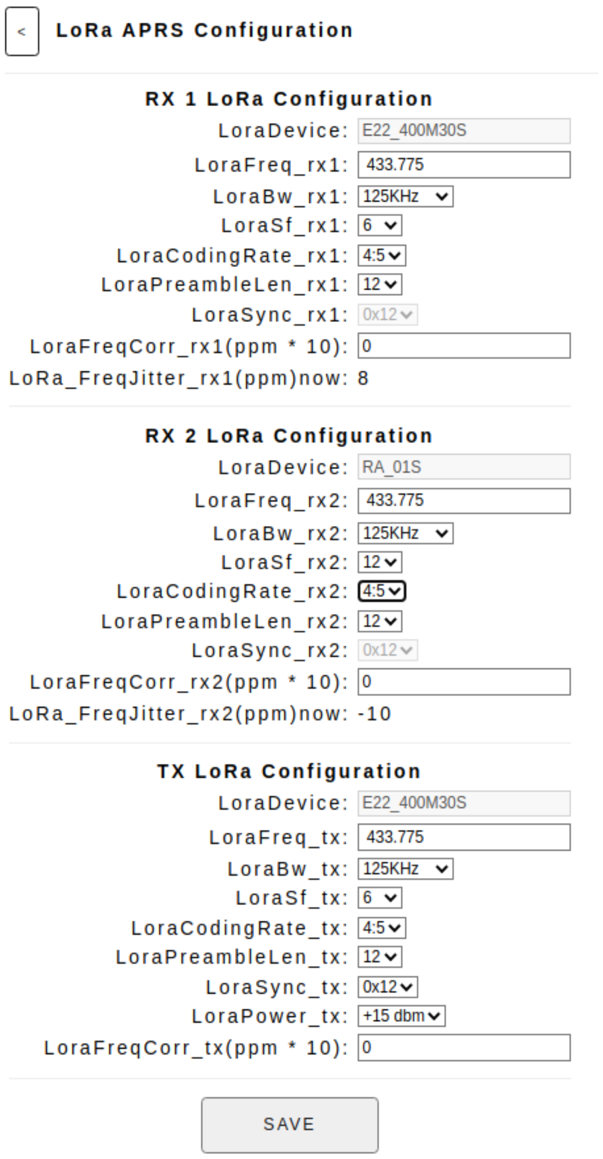
TX LoRa Configuration

LoraDevice: RA_01S
LoraFreq_tx: 433.775
LoraBw_tx: 125KHz
LoraSf_tx: 12
LoraCodingRate_tx: 4:5
LoraPreambleLen_tx: 12
LoraSync_tx: 0x12
LoraPower_tx: +22 dbm
LoraFreqCorr_tx(ppm * 10): 102

Figure 56: GUI "LoRa Configuration" page

A “LORA CONFIGURATION" GUI page is made available in order to manage this functional layer of the device; fig. 56 is the content of the this page for single radio devices, while fig. 57 is the page content for a dual-radio device.

In both cases there is a section for each RX or TX virtual device, that contain all the parameters that is possible to set for that virtual device; in particular some parameters can be set only for a device and necessarily be inherited by some others; this is for example the case of the “sync word” and the “power level”: both these parameters can be only set for the TX device but for the “sync word” will also be forced for all the RX devices.


<
LoRa APRS Configuration

RX 1 LoRa Configuration

LoraDevice:	E22_400M30S
LoraFreq_rx1:	433.775
LoraBw_rx1:	125KHz
LoraSf_rx1:	6
LoraCodingRate_rx1:	4:5
LoraPreambleLen_rx1:	12
LoraSync_rx1:	0x12
LoraFreqCorr_rx1(ppm * 10):	0
LoRa_FreqJitter_rx1(ppm)now:	8

RX 2 LoRa Configuration

LoraDevice:	RA_01S
LoraFreq_rx2:	433.775
LoraBw_rx2:	125KHz
LoraSf_rx2:	12
LoraCodingRate_rx2:	4:5
LoraPreambleLen_rx2:	12
LoraSync_rx2:	0x12
LoraFreqCorr_rx2(ppm * 10):	0
LoRa_FreqJitter_rx2(ppm)now:	-10

TX LoRa Configuration

LoraDevice:	E22_400M30S
LoraFreq_tx:	433.775
LoraBw_tx:	125KHz
LoraSf_tx:	6
LoraCodingRate_tx:	4:5
LoraPreambleLen_tx:	12
LoraSync_tx:	0x12
LoraPower_tx:	+15 dbm
LoraFreqCorr_tx(ppm * 10):	0

Figure 57: GUI "LoRa Configuration DM" page

The actual pages contains for most of the settable parameters a list of possible values to be set: the content of the presented lists will depend on the exact LoRa device associated to that device; so different modules could have a different set of possible values; in this way it is not possible to use unallowed values for the precise device in use.

In the LoRa configuration page only a subset of “high-level” parameters described in the modules programming manual for the LoRa modules can be configured; the low-level parameters are not shown because they are already optimized for the HW Platform and they should not be changed by the end-user.

Many of the parameters are intuitive and their setting depend on the type of experimentation you intend to do.

In principle, for two devices to communicate with each other, the LoRa parameters of the two devices must coincide, taking into account the link direction (i.e. RX of one end point should match TX of the other link end point).

The only parameters that can differ are the transmission power value and the frequency correction value.

This last parameter can typically be left at zero for devices that mount high frequency precision LoRa modules (e.g. the E22_400M30S model or other second generation modules equipped with TCXO) while it will be set experimentally for first generation LoRa modules which do not have good frequency accuracy or for second generation ones without TCXO. **For this purpose, the “FreqJitter” value measured in real time from the packets received and shown in the last line of the screen can be used as a suggestion.**

As a verification of the correct frequency alignment, the transmitted signal can be observed with an SDR receiver to evaluate the offset value in ppm (parts-per-million) to be set.

It is important to note that any device, also single radio ones, con operate the RX and TX virtual LoRa Devices on nearly completely different LoRa modes: this is called split-mode operation and allows to make any device compatible with either the legacy LoRa networks then the “new Generation” LoRa Networks based on the usage of dual-radio Dual-Mode iGates (using this SW ver. 5.x).
Regarding the actual LoRa Modes to use for a Dual-Mode dual-radio device it is advised to choice orthogonal LoRa modes and use for the primary radio (RX1-TX) the mode that provides the fastest equivalent channel bit rate.

A special note applies for a parameter expressly not presented in the LoRa setup page: it is the CAD (Channel Activity Detection) attribute: this funtionality, as may be already known, is present in both the first and second generation LoRa devices, but is pratically usable only with the second generation devices due to the fact that in first generation devices the CAD is only sensible to the LoRa preamble and this makes it in practice of no real benefit. With second generation devices it represents a very effective method for governing the access to the radio channel and proves to be the only practical method to implement a channel access similar to the traditional Collision Detection methods due to the fact that thanks to the processing gain of LoRa in order to discover a signal presence on the air it is necessary to perform the LoRa physical layer algorithms to detect any signal presence; the traditional collision detection method based on the signal level will not work correctly beeing possible ho have LoRa signals completely buried in the channel noise but still correctly recoverable thanks to LoRa processing gain.

In pratice the Sarimesh LoRa layer implementation allows either to send packets with or without CAD enabled, for making experimentations; but this functionality is left to API access method; when using LoRa as in the APRS or messaging applications the CAD will always be active for second generation devices and disabled for first generation devices.

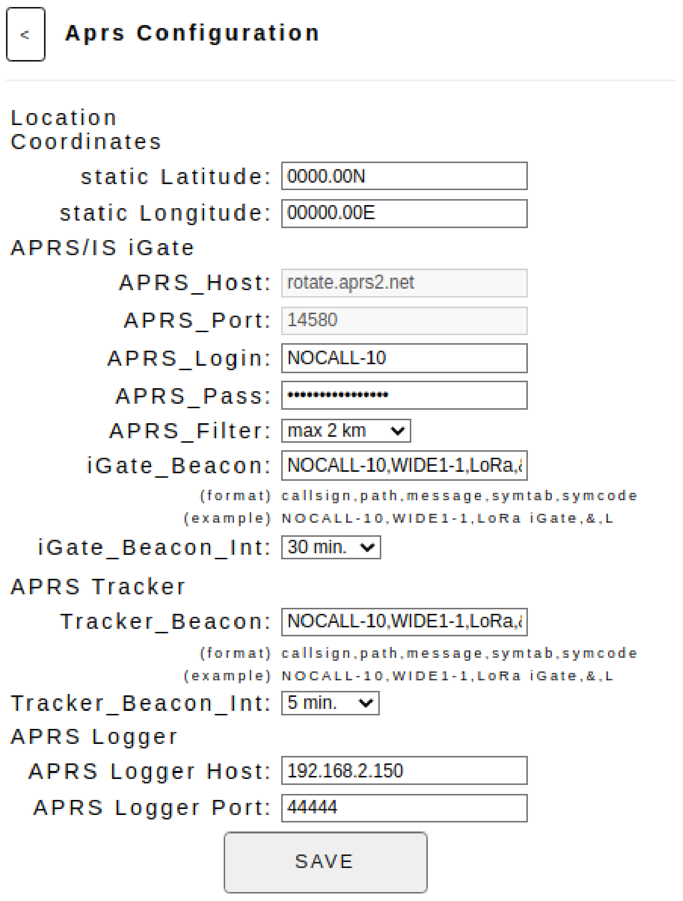
Another hidden functionality of the Sarimesh LoRa Layer is the possibility to send packets with an optional randomization of the emission times: this feature is intended to avoid that in a network consisting of several LoRa nodes emitting Acknoledgement packets as reaction to an incoming LoRa packet (i.e. for LM Messaging application) an hight correlation is happening thus leaving to an abnormal operation of the network with high CAD losses. Also this function is only usable via API and then it is managed at application layer and is not made evident in the LoRa configuration page.

A last hidden LoRa layer functionality is the possibility to send “scheduled packets” i.e. packets wich emission time is requested with an exact time stamp or time delay. Also this functionality is usable only via API and is at present used by example by the “Synchronous Beacon” application (still not documented due to experimental status).

7.4 APRS subsystem setup (any version)

The APRS is the main top level functionality of the LoRa_Tracker project: it is implementing the standard APRS functionality using as a physical layer the LoRa protocol and radio.
With respect to a traditional APRS device it differ mainly in the radio layer and is compatible at the higher layers with the traditional APRS specification and network.

To configure the APRS application there exists two different GUI pages: a first page contains all the relevant users informations related to the service, while the second page allows to customize the actual beheaveur of the device related to the APRS functions to be enabled or disabled.



< **Aprs Configuration**

Location Coordinates

static Latitude:	0000.00N
static Longitude:	00000.00E

APRS/IS iGate

APRS_Host:	rotate.aprs2.net
APRS_Port:	14580
APRS_Login:	NOCALL-10
APRS_Pass:	••••••••••••••••
APRS_Filter:	max 2 km
iGate_Beacon:	NOCALL-10,WIDE1-1,LoRa,i
(format)	callsign,path,message,symtab,symcode
(example)	NOCALL-10,WIDE1-1,LoRa iGate,&,L
iGate_Beacon_Int:	30 min.

APRS Tracker

Tracker_Beacon:	NOCALL-10,WIDE1-1,LoRa,i
(format)	callsign,path,message,symtab,symcode
(example)	NOCALL-10,WIDE1-1,LoRa iGate,&,L
Tracker_Beacon_Int:	5 min.

APRS Logger

APRS Logger Host:	192.168.2.150
APRS Logger Port:	44444

Figure 58: GUI "APRS Configuration" page

In addition there is a single entry available in the “Operation” GUI page to quickly change the actual type of operation between the “Tracker” and the “iGate” operation mode without requiring complex changes.

A major characteristic of our implementation is that there is a single SW package that can be easyly configured to operate in the two major operation modes without changing anything then a single option in the GUI of the device.

In particular the first GUI page allows to independentely specify the required parametrs for both the Tracker and iGate operating Modes.

By selecting the “**APRS CONFIGURATION** ” section from the GUI main page, you access the page that allows you to configure the main parameters for the APRS service.

Fig. 58 displays the default configuration of this page: it is not usable before a customization and the SW will refuse to send any Beacon untill this “Default Configuration” will not be customized, in order to avoid to inject wrong spots over the relevant APRS portals around.

For the iGate mode, the device must first of all be able to connect to the Internet and to a suitable ARPS-IS server.

If the internet gateway in use in the Wi-Fi/ethernet upstream network is protected by a Firewall, it must be configured to allow outgoing traffic on the IP port used by the APRS-IS server and which will depend on the chosen APRS-IS server.

Then , a Login and a Password will be needed for connecting to the APRS-IS server. The password is a numeric code that can be calculated starting from the Callsign at this link: https://www.iz3mez.it/aprs-passcode/

The iGate position can be automatically acquired , if there is a GPS module installed on the board, which has reached a 3D fix; in any case a static position can be manually set in this page ; this allow to avoid a GPS to be equipped on the board for those cases where the device position is fixed and no GPS time reference in required., thus lowering the device cost.

To determine the value of the coordinates to be reported, the native APRS format must be used; that is to say:

- latitude: 2 digits for degrees + 4 digits with dot after the first two digits for first and second seconds as required by APRS followed by the letters N/S for north or south.

- longitude: 3 digits for degrees + 4 digits with dot after the first two digits for first and second seconds as required by APRS followed by the letters E/W for north or south.

Then for the iGate functionality, a parameter must be inserted that allows filtering the beacons to be transmitted on the LoRa RF network and coming from APRS-IS: the only mode supported by the GUI is the "maximum distance from the iGate position".

Note: the LoRA Channel could easily saturate if wrong settings are used! Keep tx rate as low as possible and limit the distance range to the necessary

At this point, is possible to enter the required parameters to create the beacon string to be transmitted to the LoRa network and to the APRS/IS network.

This string will be automatically composed on the basis of the parameters entered in the GUI , as illustrated in the example below:

parameters entered in the iGate_Beacon or Tracker_Beacon field:
I1XYZ-10, WIDE1-1, LoRa, &, L

resulting string beacon (location field in plain text):
I1XYZ-10 >APLS01, **WIDE1-1** :!GPS_LAT **L** GPS_LON **&** LoRa -32B125S12C8P10

resulting string beacon (compressed location field):
I1XYZ-10 >APLS01, **WIDE1-1** :! **L** 9 w?#R-GG **&** !!G LoRa -32B125S12C8P10

The “**GPS_LAT**” and “**GPS_LON**” strings indicate respectively the latitude and longitude acquired by the GPS module, or the values entered in the latitude/longitude fields of the screen in the “APRS Configuration” if no GPS module is installed.
The beacon format is compliant to the actual APRS specifications. The only fields that can be set by the user are the red ones.

The string “*APLS01*” is the “destination” field of APRS packet and identifies a Sarimesh Device. The list of all sources is available at http://www.aprs.org/aprs11/tocalls.txt .

The field following the symbol ” **!** ” represents the location of the device: if it begins with a numeric character, it is assumed that the location is in uncompressed format, otherwise it is assumed that the 13 characters that follow represent the location in a compressed format; the compression algorithm is always described in the APRS protocol specification.

With the SW 4.x and 5.x version, the software decodes locations in both formats (compressed and uncompressed), while in transmission it is possible to indicate the method to use in the second GUI page related to the APRS service; the use of the compressed mode allows to reduce the length of the location field to the advantage of a higher transmission speed of the single packet.

The characters “L” and “&” present in the above example can be replaced with other characters in order to modify the symbol with which the APRS icon will appear on aprs.fi and similar websites; for a complete list of icons, it is possible to consult the following URL: https://www.iz3mez.it/aprs-server/simboli-aprs/ (thanks to Giovanni IZ0CZW for the report). It is suggested not to change the character L as for standard use it indicates a LoRa device.

The last part of the beacon is automatically generated by the SW and summarizes the working conditions of the device, if this addition is enabled via the second APRS GUI page (see later); in particular the meaning is the following:

- working conditions (example):

-32B125S12C8P10

- meaning of the fields if present:

"**-**" indicates that the beacon was generated in a blacklisted area (for testing only)

32	Sequence Number "module 256" for packet (it is incremented each beacon package generated)
B125	Bandwidth used for transmission in Khz (without decimals)
S12	Spreading Factor used for transmission
C8	Coding Rate used for transmission
P10	Preamble Length (number of symbols)

In the generated beacon strings, there may also be two other fields enclosed by parentheses containing one or more subfields which reports the first and last node traversed by the APRS packet; this easily allows to have an indication (in particular for the second case) of the receiving condition of the packet by the node indicated in the first subfield of the parenthesis: the meaning can be deduced from the following example:

- device field traversed:

(IQ8SO-10 -120 -16 186)

- subfield decoding:

IQ8SO-10	callsign of the traversed node
-120	signal strength of the spot received from the traversed node in db without decimals
-16	SNR of the spot received from the traversed node in db without decimals
186	frequency offsett measured by the traversed node receiver in Hz

These optional fields are included in the comment part of the beacon and are clearly NOT APRS standard. They can be partially or completely omitted in order to have packets issued in compliance with the APRS standard; it can be enabled or disabled via the second APRS GUI page.

In order to optimize the channel occupation, the device will transmit the optional fields only every 5° packet of a sequence of one hundred packets. For all other packets in a sequence of one hundred, only the sequence number will be transmitted. The function of the sequence number is to be able to highlight packet loss situations; it is obviously useful only in the experimentation phase to deepen the analysis of the behavior of the LoRa system.

A further option that can be enabled/disabled via the second APRS GUI page, is the inclusion in the Beacon payload of the weather conditions acquired via the locally installed weather sensor: in this case a standard weather extension package is inserted in the actual beacon payload according the standard format defined by the APRS specification.

A key aspect to be specified is then the "rate" at which the beacons should be generated. Both the Tracker and the iGate modes allows to use several different sending rate strategies in order to implement different usage patterns.

Let's spend some words about the beacon interval algorythm. The function of beacons is obviously to report the status of the device that transmits the beacon: the nature of this data can be very different and generally includes a "location" as the main data, (i.e. an identifiable position or via geo coordinates or through a QRA locator); in some cases it is also possible to avoid transmitting this basic data because it is not subject to change (as for example for a weather station or for a fixed iGate); in these cases the data is generally transmitted at bigger time intervals, avoiding the transmission of data that does not change over time.

In the classic use of APRS, the main aim is to highlight the movement of the device over a territory, so it makes sense to transmit beacons only in case of position change, in the case of LoRa experiments the "radio mapping" aspect of the territory becomes interesting and important in order to evaluate the radio coverage reached thanks to LoRa protocol.

In this last situation it therefore becomes interesting to be able to issue beacons with a logic linked precisely to the use of the collected data... for example. use the change of position or the path followed by the mobile device as a reference basis for the emission of the spots: an example is to detect the radio coverage map of a certain area or the level of coverage of a certain route (eg a mountain route or a particular road).

The "Agile Beaconing" function aims to dose the beacon emission times according to various criteria in order to obtain the above objective.

In the specific case, the data acquired in real time by the GPS (in particular position, direction of travel, time, speed and altitude) are used to appropriately dose the emission times of the beacons.

There are two main sub-modalities: modalities based simply on the distance traveled or modalities based on "events", meaning by this term the sets of predefined and significant conditions of the path being followed.

It is then possible to completely disable the Agile function, in this case falling back to the basic beacon emission mode, i.e. the constant time mode; in any case, even if the "Agile" mode is active, the time parameter indicated in the APRS function setup screen remains active so that beacons can be sent at worst every certain time if there are no events.

It should be noted that given the nature of positioning data derived from GPS, it is normal for such data to be affected by an error in particular for the position which can be evaluated in a few meters and which depends on the GPS signal reception conditions: to avoid excessive sending of beacons due to these errors it is however avoided to transmit position data that differ by a few meters.

A similar limitation exists for the time interval between beacons: in this case, to avoid situations of congestion, transmissions of beacons that are too close in time to previous beacons are avoided.

In any case, it is always possible to immediately send a beacon manually by shorth pressing the button available on the devices.

A further function linked to beaconing is to avoid issuing beacons when the mobile phone is in particular areas from which you want to avoid transmitting data for privacy or technical reasons, e.g. to avoid radio congestion problems.

For this purpose it is possible to define a set of areas, identifiable by means of a center and a radius, such that if one is within one of these areas the beacon is suppressed.

Several beaconing strategies are then implemented: “time based ” , “space based” and “event based”.

The default strategy is the “time based” static beaconing: in this strategy, that can be setup via his first APRS GUI page, the emission rate of the beacons is always the same during the device operation and just can be different between Tracker or iGate operation mode. A number od different rates can be preset and the rate does not change in any way during the operation time.

This default strategy is always active and represents the last resort the device should have for giving any sign of life...

Via the second APRS GUI page a second group of strategies can be enabled: they are called “agile beaconing” and target specific use cases mainly for a tracker.

The reasoning for having these other strategies is in order to perform specific uses for a tracker, in the framework of LoRa experimentation: they allow to perform a sort of “mapping” activities as described before: that is to perform a detailed acqusition of LoRa radio coverage data in a specific area to better study the LoRa operation and make possible to compare different usage strategies for its modes.

In details the "**Agile Beaconing**” item allows to set the algorythm for the optimization of beacon transmitting rate considering the trajectory of the moving tracker. The implemented algorithm can be set at different strategies in order to achive different goals; the available modes are the following:

1. **Disabled**: no agile beacon active; beacon will be generated according to the infos provided in the APRS main GUI page (i.e. static operation based on time). This is the normal operation mode and performs time based beaconing.
2. **EventMode**: the device will send a new beacon when an “event” is detected, like a change od direction, or the different speed of the mobile, or the stop/start of movement. Suitable thresholds are set to optimize this mode as a “**detailed radio coverage discovery**” mechanism. Not to be used for normal operation.
3. **MappingModeXXX**; it is intended for creating a “**mapping**” of the present area where the device is , by sending a beacon based on the “**distance**” the mobile moved; **XXX** can assume following values: **UltraFine, Detailed, Course, UltraCourse.** Also this modes are not intended for normal operation, but for “**radio coverage discovery**”.

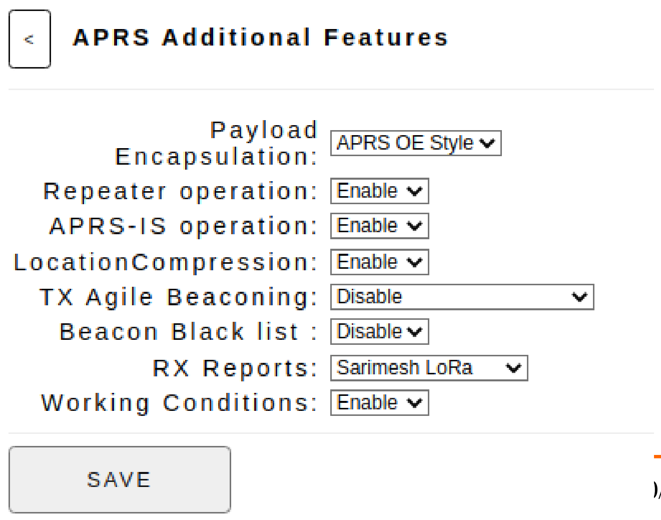
A further beaconing function available is the “**Beacon Black List**”: this function allow to exclude a set of geographical "zones" from sending beacons, mainly for privacy purposes or to mitigate any radio congestion problems in particular areas such as eg. a town centre. This feature is not jet supported by a GUI page at present, but will be supported ASP .

The actual way to specify a “BlackListed” area is by providing a centerposition coordinates and a radius via a suitable table; at present what is missing is the GUI to easyly specify this table content; it can already be specified via the SW build definition File.

A last function that can be enabled from the main APRS GUI page is a “**Logger Server**”. The device is able to connect to a server dedicated to collect data related to the APRS spots received from the LoRa network; this interface is based on a simple UDP protocol and allows to show spots on a geolocated map.

The documentation of this interface is not jet available due to too experimental status of this feature. Basically this interface is taylored upon the “direwolf” logging style for the APRS received traffic by an iGate and allows to send to a dedicated server all and every spot detailed content in order to store it in a suitable database for further processing; it is a very important tool in order to capture usage data during a possible mappimg campaign for creating any sort of radio coverage images, or to create historical geolocalized maps of radio coverage.

This server can be located anyway over the internet and do not depend on any third party service like aprs.fi . It requires an upstream internet connection for the reporting node that is typically an iGate that collects beacon spots coming from one or more mobile trackers.



Payload Encapsulation:	APRS OE Style
Repeater operation:	Enable
APRS-IS operation:	Enable
LocationCompression:	Enable
TX Agile Beaconing:	Disable
Beacon Black list :	Disable
RX Reports:	Sarimesh LoRa
Working Conditions:	Enable

Figure 59: APRS additional Features

To complete the APRS layer programming the second APRS GUI page “**APRS FEATURES**” has to be described; fig. 59 contains the available attributes that this pages allows to set.

We have already described most the option already during the previous paragraphs, so just need to add few details for the not mentioned items here.

The first option is to specify which type of payload encapsulation the device should use for beacon transmission: the traditional APRS payload as specified in the APRS specification was using an AX.25 encapsulation mode in order to implemented the expected link-layer APRS capabilities.

This encapsulation mode has been used for so many years and is used by the totality of APRS devices available on the market till now.

With the advent of LoRa as a physical layer , thanks to its attribute of “binary transparency” it was clear that the AX.25 encapsulation was nearly redundant for several reasons and was incresing the ProtocolDataUnit (PDU) size at the expense of available radio channel troughput: so few years ago, mostly from german and austrian radio amateurs a simplification was introduced in the way a PDU could be identified, just taking benefit that beeing the lowest LoRa layer already a packet layer with included mechanisms for PDU delineation , error detection and correction. Then a very simple mechanis was introduced in the for of a few caracter header just to be able to discriminate different types of payload. We define this type of pseudo-encapsulation OE-Style in our wording.

So at present most of the available LoRa APRS implementations are using this type of simplified encapsulation. Unfortunatelly it makes the actual interworking with the old traditional TNC suboptimal.

In order to have an optimized mechanism for interaction with the existing installed base of traditional APRS devuces a possible alternatives is to just use the original AX.25 encapsulation method at the xpense of a little incresead PDU size.

Our LoRa APRS implementation is able to receive APRS PDU with both the OE-Style and the AX.25 encapsulation method thus performing the required operation at the fly to adapt the actual PDU to the exactly same format when relaying those PDU toward the internet via APRS-IS or over the LoRa network while implementing repeater operation.

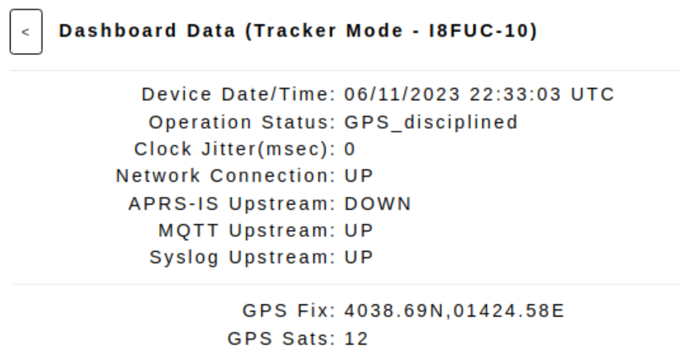
The Encapsulation option available in this GUI page allows to set the encapsulation style to use by our device when trasmitting any PDU toward the APRS-IS or the LoRa networks.

The options that follow on the GUI page are the “repeater Operation” and the APRS-IS operation: these are the two classical option available on old APRS devices and enable/disable the digipeating over the LoRa network side for the received beacon over that network, and the reception/sending of APRS spots from and to the internet APRS-IS network.

These options can be used to deep customize the iGate operation for specific use cases like having an iGate operational in a site where no internet connectivity is available, or if we do not want to operate the APRS-IS connection at all.

The remaining options have been already discussed in the previous part of this chapter.

7.5 Dashboard screen (any version)



```
< Dashboard Data (Tracker Mode - I8FUC-10)

     Device Date/Time: 06/11/2023 22:33:03 UTC
     Operation Status: GPS_disciplined
    Clock Jitter(msec): 0
   Network Connection: UP
     APRS-IS Upstream: DOWN
        MQTT Upstream: UP
      Syslog Upstream: UP

              GPS Fix: 4038.69N,01424.58E
             GPS Sats: 12
```

Figure 60: GUI "Dashboard" page showing GPS fix

The Dashboard allows to have a general overview of the device working conditions, from an operational point of view and from a configuration point of view.

Fig 60 is the first section of the Dashboard and has been already described before.

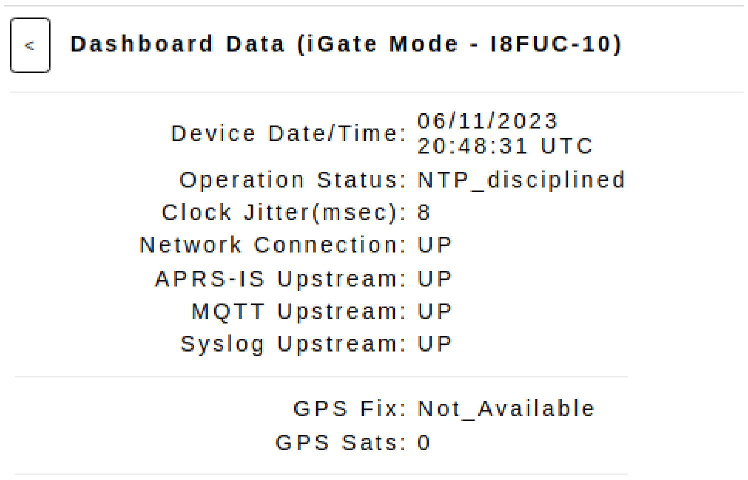
Fig 60 shows the case of a device disciplined by the GPS that has acquired a full 3D-Fix with 12 Satellites used for the fix... this is a very nice fix!

The Clock Jitter value in msec is also reported, i.e. an indication of how much the local time varies over time in relation to the type of synchronization in progress. With a GPS disciplined situation this value is nearly always 0 msec.

The same section provides the status of the main networking services: the Network connection is actually the status of the upstream internet connection that can be either via WiFi or via Ethernet LAN connetion.

The APRS-IS Upstream connection is actually the connection with the APRS-IS server: in this foto due to the fact that the device is operating as a Tracker, the APRS-IS connection id down.

The MQTT and Syslog upstream appears to be up: this is consistent with the fact that the tracker is presently in my lab and is connected to internet via the WiFi upstream connection.



```
<  Dashboard Data (iGate Mode - I8FUC-10)

    Device Date/Time: 06/11/2023
                      20:48:31 UTC
    Operation Status: NTP_disciplined
   Clock Jitter(msec): 8
  Network Connection: UP
     APRS-IS Upstream: UP
        MQTT Upstream: UP
      Syslog Upstream: UP

              GPS Fix: Not_Available
             GPS Sats: 0
```

Figure 61 : GUI "Dashboard" page showing NTP disciplined

Fig 61 shows the case where the actual device has been set as an iGate, and the GPS has been disconnected.

This section also lets you know in which operating mode the device is presently operating; from the example above, the device is now in iGate mode.

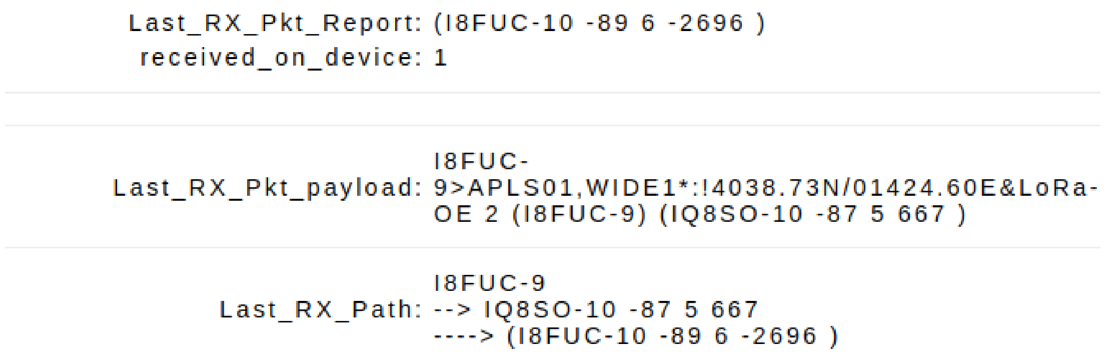
In this case eg. it can be seen that the GPS is not active and therefore the local time is derived from the NTP internet protocol; as a result it is highlighted that the local clock has a jitter of a few msec as is normal for the time derived from the internet.

The Jitter value is important for some features, described in the next chapters, that require a pseudo-synchronous timebase for their operation.

The same figure now show that the APRS-ISupstream connection is up , consistent with iGate role to be connected to an internet located APRS-IS server.

A next section of the DashBoard, shown in 0g. 60, summarizes the status of the receiving part of the device.

In particular, this section reports only the last packet received and for this packet it shows:



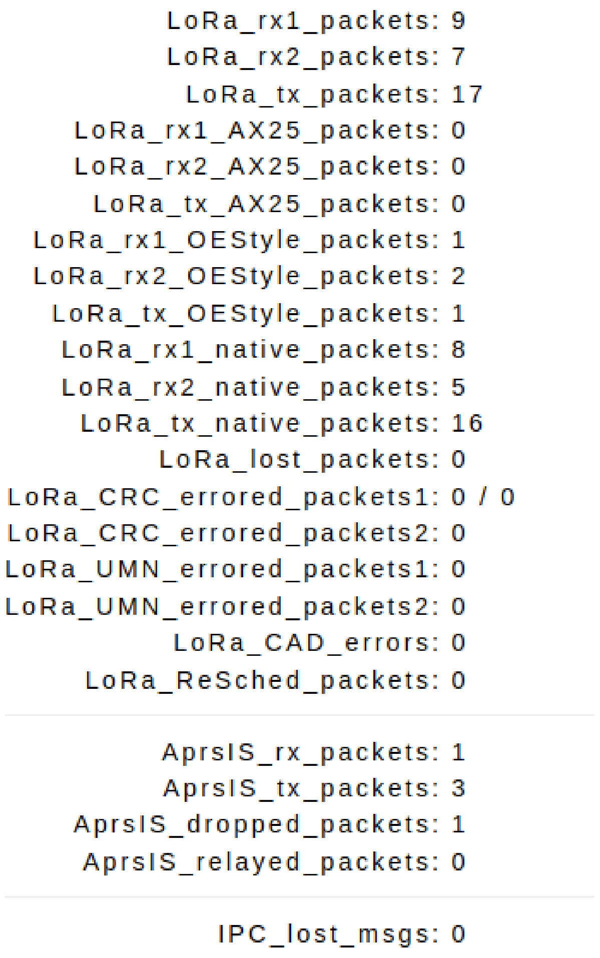
```
Last_RX_Pkt_Report: (I8FUC-10 -89 6 -2696 )
 received_on_device: 1

                    I8FUC-
Last_RX_Pkt_payload: 9>APLS01,WIDE1*:!4038.73N/01424.60E&LoRa-
                    OE 2 (I8FUC-9) (IQ8SO-10 -87 5 667 )

                    I8FUC-9
       Last_RX_Path: --> IQ8SO-10 -87 5 667
                    ----> (I8FUC-10 -89 6 -2696 )
```

Figure 62 : Dashboard showing "last packet" information

- the "**reception report**" that our node have created for the received packet and that would be attached to the packet it it will be didipeated or transmitted to APRS-IS: this record contains the identity of the node that is receiving the packet, the received signal level in dBm, the signal/noise ratio in dB and the frequency offset in Hz.
- the payload contained in the received packet: if the packet is received as "CRC Errored" it means that the packet, despite having been recovered, is affected by uncorrectable errors which may contain incorrect characters or unprintable symbols; the strategy is to report the recovered content in any case for any subsequent actions. In case of CRC errored packets no digipeating or APRS-IS forwarding will take place.
- path of the received packet: only the identifiable components of the path are reported, i.e. usually only the original source of the received packet (regardless of any repetitions that the packet has been subjected to) and the last node the packet passed through.

The next section of the Dashboard, shown in 0g. 61 , will report a series of counters related to the traffic of packets passing through the node; the meaning of the various counters is quite intuitive based on its name.

In particular, the packets are listed according to their encapsulation (AX_25 or OE_Style), according to the direction in which they have been seen (TX or RX from the point of view of the node), according to the type of recognized payload (APRS or native format, described in 7.7); the lost packets are then reported (due to unavailability of the channel being transmitted) and the packets received but revealed to be CRC_Errored, ie affected by uncorrectable transmission errors.



```
          LoRa_rx1_packets: 9
          LoRa_rx2_packets: 7
           LoRa_tx_packets: 17
     LoRa_rx1_AX25_packets: 0
     LoRa_rx2_AX25_packets: 0
      LoRa_tx_AX25_packets: 0
  LoRa_rx1_OEStyle_packets: 1
  LoRa_rx2_OEStyle_packets: 2
   LoRa_tx_OEStyle_packets: 1
   LoRa_rx1_native_packets: 8
   LoRa_rx2_native_packets: 5
    LoRa_tx_native_packets: 16
         LoRa_lost_packets: 0
LoRa_CRC_errored_packets1: 0 / 0
 LoRa_CRC_errored_packets2: 0
 LoRa_UMN_errored_packets1: 0
 LoRa_UMN_errored_packets2: 0
           LoRa_CAD_errors: 0
     LoRa_ReSched_packets: 0

         AprsIS_rx_packets: 1
         AprsIS_tx_packets: 3
    AprsIS_dropped_packets: 1
    AprsIS_relayed_packets: 0

             IPC_lost_msgs: 0
```

Figure 63: GUI Dashboard statistics

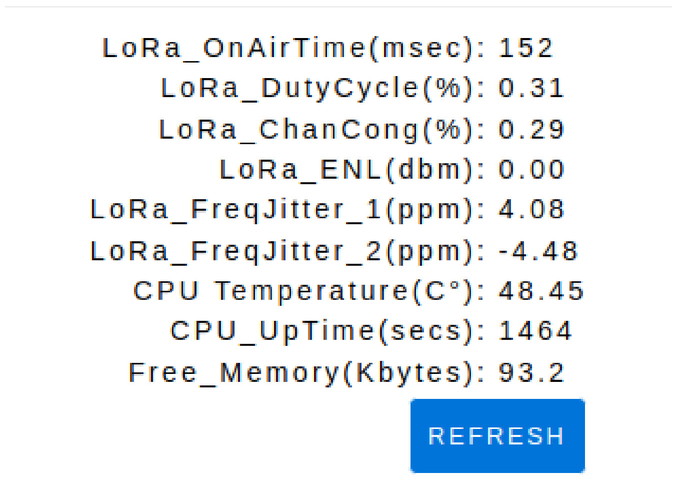
The packets that failed to transmit due to the occupation of the radio channel revealed by the CAD (Channel Activity Detection) mechanism are then reported; a further parameter is the number of packets rescheduled in the transmission phase again due to radio channel occupation problems.

From APRS-IS side (APRS connection to and from the internet) the packets received and transmitted are reported as well as those repeated towards the LoRa network or suppressed as they do not comply with the repetition policies towards the LoRa network side.

The last parameter reported is the number of messages lost in the interprocess-communication (IPC) system used internally by the SW to make the various components of the SW interact with each other: this parameter is an indicator of possible SW congestion.

It should be noted that the counter of messages received from the APRS-IS side takes into account the filtering applied in that direction by the node: in particular, the only filter applied is the distance filter (intended as distance between the transmitted beacon location and the iGate position).

The last section of the Dashboard shows a series of data relating to the node in its entirety; in particular the first parameter shows the moving average of the transmission time values (OnAir-Time in msec) of the transmitted packets, the DutyCycle in % of channel occupation or how much of the transmission time of the node has been actually used by the node, and a further parameter (heuristic) which gives an approximate indication of the level of congestion of the radio channel as perceived by the node).



```
LoRa_OnAirTime(msec): 152
   LoRa_DutyCycle(%): 0.31
    LoRa_ChanCong(%): 0.29
      LoRa_ENL(dbm): 0.00
LoRa_FreqJitter_1(ppm): 4.08
LoRa_FreqJitter_2(ppm): -4.48
 CPU Temperature(C°): 48.45
   CPU_UpTime(secs): 1464
 Free_Memory(Kbytes): 93.2
```

Figure 64 : GUI Dashboard statistics - part 2

Two data relating to the LoRa level follow: the value of "ExstimatedNoiseLevel" (ENL in dbm) evaluated for the site where the node is installed and the moving average of the Frequency Difference values measured on the received packets.

The ENL value is a heuristic estimate of the noise level in dbm of the site calculated considering only packets received with a negative signal-to-noise ratio (SNR): therefore this parameter will be present only if the node has received packets with a negative SNR. It represents the dbm value (evaluated by the lora chip in its internal operation) of the signal intensity at the LoRa chipset terminals regardless of the "process gain" values that the LoRa protocol allows to obtain.

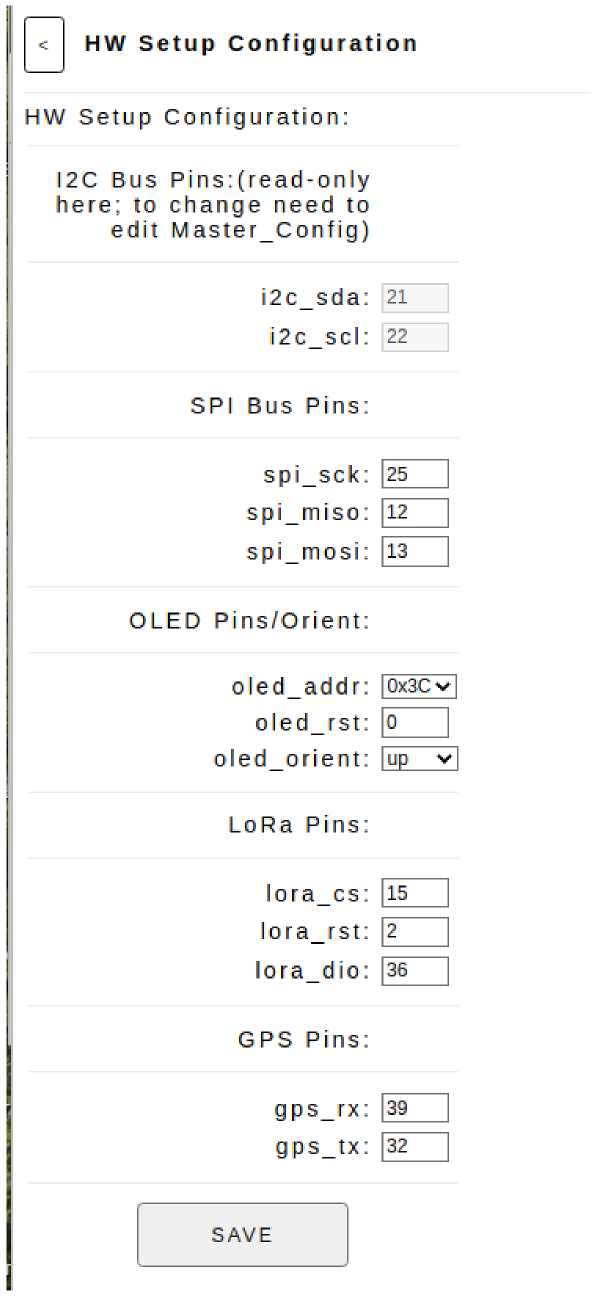
The indicated Jitter value (in parts-per-million) is the moving average of the frequency difference values divided by the value of the transmission frequency that the node was able to measure on the received packets regardless of where these packets were transmitted : therefore it represents a form of estimation of how much the current node could be "retuned in a frequency" to operate better in the LoRa network in which it operates.; this value is presented in the node's LoRa setup screen to be effectively used as a correction value to be applied to the LoRa frequency setup.

It is to be noted that for dual-radio devices, all the RX related items are reported individually , so that it is possible to discern how the two radios are operating.

The last values presented are an estimate of the temperature value of the node CPU and the value of the operating time (in sec.) since the last reboot of the node itself, an estimate of the actual FreeMemory that the processor reports and the device uptime from the last reboot.

7.6 HW Setup Configuration

With 5.x version of the SW, the use of the Sarimesh SW on HW platforms other than the Sarimesh one is supported with few limitations related to the HW architecture of the target devices.


HW Setup Configuration

HW Setup Configuration:

I2C Bus Pins:(read-only here; to change need to edit Master_Config)

i2c_sda: 21
i2c_scl: 22

SPI Bus Pins:

spi_sck: 25
spi_miso: 12
spi_mosi: 13

OLED Pins/Orient:

oled_addr: 0x3C
oled_rst: 0
oled_orient: up

LoRa Pins:

lora_cs: 15
lora_rst: 2
lora_dio: 36

GPS Pins:

gps_rx: 39
gps_tx: 32

Figure 65: GUI "HW Setup Adjustment" page

Since the SW is very modular and based on standard HW components, it is easy to adapt the code to run on almost all boards based on the ESP32 processor and which have the necessary HW components (e.g. a GPS module and a LoRa chipset) .

If the SW is used on HW platforms other than SARIMESH platforms, in general it may be necessary to appropriately set some features of the SW in order to adapt to the different characteristics of the HW; in particular what may differ, within a certain type of HW card, is the value of some pins of the processor to which the peripherals present on the card are connected.

The GUI page shown in 0g.63 allows you to set the pins of the ESP32 processor to be used for the following functions:

- I2C bus (*)
- SPI bus
- OLED pins, addr and orientation
- LoRa specific pins
- GPS specific pins

(*) For the I2C bus it is necessary to modify the SW build definition file. So a separate SW image will be distributed.

The values to be set can be found in the hardware documentation of the devices used.

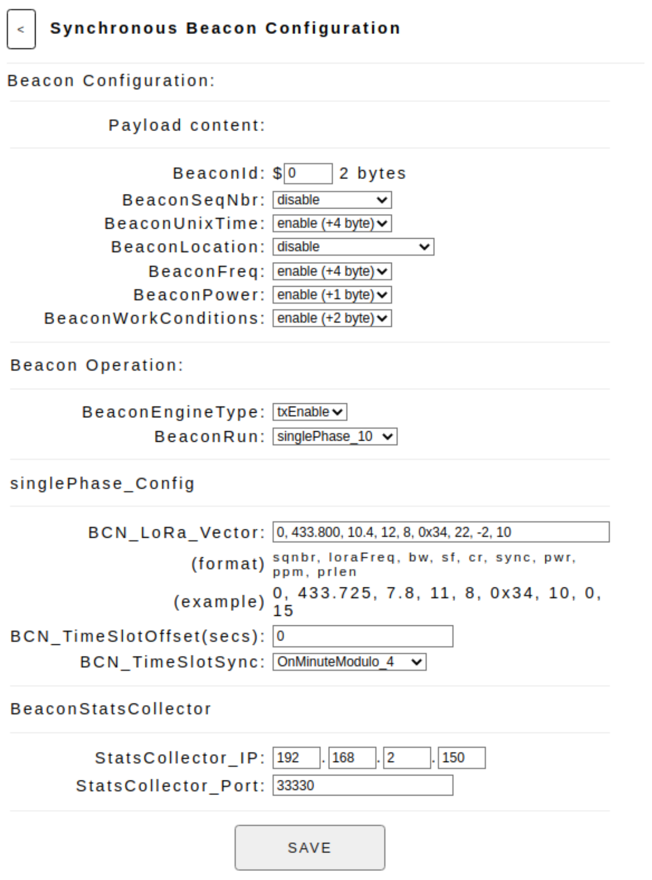
For the SARIMESH HW this page shows the values used but not modifiable as they are invariant.

Based on the preset status of the TTGO like HW variant it seems that the I2C pin used is nearly a stable choice; so this makes possible to adjust HW incompatibilities via this page without any further action; in pratice the TTGO like device should startup and be able to display the GUI also if some pins are not correctly set; the via this page it should be possible to adapt the SW to the preset HW configuration.

Another variant of the presently available TTGO devices is the power control chip installed... at present the SW Vr. 5.x supports the AXP192 chip; new chips may need to be checked (not done due to missing HW samples to test it).

7.7 Synchronous Beaconing Subsystem Setup (any version)

The Sarimesh SW Sarimesh contains, starting from version 3.x , an undocumented feature that we defined "Native Beaconing" and now redefine as "Synchronous Beaconing": it is an experimental function that operates the node in different modes with a static "time slicing" type approach.



< Synchronous Beacon Configuration

Beacon Configuration:

Payload content:

BeaconId:	$ 0 2 bytes
BeaconSeqNbr:	disable
BeaconUnixTime:	enable (+4 byte)
BeaconLocation:	disable
BeaconFreq:	enable (+4 byte)
BeaconPower:	enable (+1 byte)
BeaconWorkConditions:	enable (+2 byte)

Beacon Operation:

BeaconEngineType:	txEnable
BeaconRun:	singlePhase_10

singlePhase_Config

BCN_LoRa_Vector:	0, 433.800, 10.4, 12, 8, 0x34, 22, -2, 10
(format)	sqnbr, loraFreq, bw, sf, cr, sync, pwr, ppm, prlen
(example)	0, 433.725, 7.8, 11, 8, 0x34, 10, 0, 15
BCN_TimeSlotOffset(secs):	0
BCN_TimeSlotSync:	OnMinuteModulo_4

BeaconStatsCollector

StatsCollector_IP:	192 . 168 . 2 . 150
StatsCollector_Port:	33330

SAVE

Figure 66: GUI "Synch Beacon Configuration" page

First of all, this function assumes that a node operates in a GPS_Disciplined or even NTP_discipled mode in such a way that the local time values are sufficiently "precise" , that is having tolerances within a maximum of a few tens of msec.

In practice, the operating time of the node is managed dynamically on the basis of a certain "usage schedule" consisting of a certain time base (e.g. 3 minutes) aligned in a certain way (e.g. on a time base of UTC clock): this program specifies how to operate in the subintervals of this program duration time (slice time).

In each time slice, the node will be able to operate in a completely different settings, i.e. with different frequency values, emission mode and power as well as packet encapsulation typology and relative payload.

An example that has been used for about a year in a particular experiment carried out on a 120 km radio section and documented on the Sarimesh.net website (http://www.sarimesh.net/lora-beacon-propagation-test/ and http ://www.sarimesh.net/lora-propagation-test-bed-2/).

The experiment was based on a pair of nodes operated in Native mode for 100 sec for simulate very marginal conditions of the link (**SF12 / BW 10.4 Khz**) and for the remaining time (80 seconds) in APRS mode with the standard parameters (**SF12 / BW 125 Khz**) on a different frequency.

The objective was to acquire and compare the typical parameters of the connection in the two different operating modes.

For setting the mixed operating mode, a dedicated page has been prepared in the GUI which is illustrated in fig. 66.

To activate the mixed operation function it is sufficient to activate the Syncronous Beacon function in the "Operation Mode" page.

A "native" default payload has been defined in the GUI. It contains all the data selected in the first section of the screen and arranged in very few bytes; to identify a node, a byte with a numerical code is simply used; the maximum number of active nodes in a given experiment depends on the slicing program defined below.

The coding of the various parameters is defined in the SW and for those interested just go and consult the sources of the SW.

The "Beacon Operation" section follows which specifies the type of program and whether the various phases are reception-only or bi-directional.

The "BeaconRun" item defines the various types of use of the program time and can assume SinglePhase values (of varying duration from 18 sec to 100 sec) or tuning mode in which the beacon automatically follows a sequence of transmission/reception phases with different frequency values to find out any frequency shift).

Then follows a section that specifies how the Native Beacon phase takes place: in practice it synthetically specifies the working conditions at the LoRa level to be used and the time offset with respect to the program time from which to start with the native beacon phase.

The APRS type phase is defined to complement the program duration time and uses the LoRa / APRS parameters of the node as described in the relevant screens.

To ensure the coexistence of several nodes in an experiment phase, the "Identity of the node" parameter is exploited with the simple logic that in a certain time interval only a certain node can transmit; in particular, the NodeId parameter defined above is used as an index to activate the transmission phase of only one of the nodes in a fraction of the time assigned to the "native beacon" phase of the program; this operating mode is only available by selecting a duration of the native beacon phase of 100 sec.; for all other values a single pair of nodes is always assumed as the target of the experiment.

The function described above is, as can be seen, still very experimental and represents only a "proof of concept" to be possibly better defined and exploited in the future.

8 Device configuration Management

The Sarimesh Core SW that is the generic SW platform we use as a base for the LoRa Tracker SW builds has been designed to be used for making devices able to be managed by inexpert users not by experienced SW designers; so it assumes that to make any device configuration it must not be required to use any special SW development environment, but just use features available on the device as it is.

This is the de facto standard for any embedded device presently available on the market as for example WiFi devices, ADSL routers, etc.

The approad we have chosen is exactly according to the defacto standard in terms of management of such devices.

Up to SW vr. 4.x there was some divergencies we have removed with the last SW vr. 5.x, by implementing a reworking of the GUI and device configuration method.

Till SW ver. 4.x the device configuration was performed via a GUI interface that was managing and modifying a number of parameters internally keept in a fast FRAM (for Sarimesh HW devices) and in slow and weareable flash (via a Little FS Filesystem) for TTGO T-Beam and similar devices.

This was a major difference between Sarimesh and non Sarimesh HW devices and was implying a frequent access to the on processor flash memory that would slow the GUI operation and wear the flash for non Sarimesh devices.

With SW Vr. 5 there is now an improvement of the configuration method that should solve the slowing and frequent access to the flash and give GUI speed for the TTGO similar to the Sarimesh HW that incorporate a very fast FRAM .

In pratice any configuration session for a device may include the access to few pages of the GUI that modify a number of parameters; in order to make a complete configuration several pages may be required to be modified; all these modifications will only impact an “in RAM memory” status of a “temporary configuration file” in JSON format that keeps normally the running device configuration.

Any change to this “temporary configuration” will have immediate effect for the few “immediate action” parametrs (as for ex. the debug parameters) but will not be reflected in the “permanent configuration” keept in non volatile memory; in order to freeze this temporary configuration in the nonvolatile memory a “commit” operation will be required: this explicit operation will allow to set the present temporary configuration as the running configuration after the next device reboot.

This method allows to discard , by not committing , any partial configuration change should the user decide to abort the re-configuration session; a reboot, without a previous committ, will restore the configuration to the last committed state.

The commit operation actually will just copy the “in RAM memory” JSON configuration file to the FRAM or to the flash (depending on the HW device) in a one-shot operation; all the intermediate configuration changes will not impact the flash in any case.

This approad the allows to speedup the configuration screens operations and will reduce drastically the accesses to the on processor flash memory fot the devices that do not have a FRAM for keeping the device configuration.

A further benefit of this approach is that any change/save of device configuration via Configuration file upload/download no longer requires to go trought the “Admin mode” step: till SW vr. 4.x this operations was requiring to put the device in a special “Admin status” and then would imply a double reboot for the device in order to save or restore a configuration file; now this step is no longer required ad a configuration can be saved or restored on/from file immediately; for restore operation from a configuration file, after the file upload a commit operation is required to make the new configuration the active one, after the next reboot.

The actual configuration of a device is then a single file in a JSON format: then it is very easy to manage such file by using any SW tool able to display/modify JSON files.

This approad allows to save and restore any device configuration in order to keep a clear track of any configuration variant used during any test session if required.

A further advice is here presented regarding the “factory default configuration”: typically any device when received by the end user should be able to be powered-up and be immediately usable for its tasks: with SW vr. 5.x we make this possible by incorporating in the SW builds we distribute a “factory defaults restore” feature that let a new device discover during the power-up if the device holds a valid configuration, and just in case a default configuration, keept in the SW image, is forced and made active; in this way any device should be able to come-up after a reboot with a valid configuration that allows to perform the necessary tuning of the device only using the standard GUI available on the device.

The present factory default configuration is to let the device come up with only the essential features enabled and as an APRS tracker with dummy user related informations; the default configuration will be detected as such during the startup and a suitable message will be presented on the local display to warn the user in order to let him finalize the device configuration with his personal details.

A further function available regarding the configuration is the possibility to “restore the default factory configuration”: this operation could be required to escape from any unexpected configuration issue and restore normal device operability.

The major unexpected configuration issue could be the case when a device running a given SW image version is upgraded to a new SW release: infact any new SW realease at present do not take care of any migration of the existing configuration of a device to the may be update configuration required by the new SW release.

If this is the case a device that has been updated could not be able to reboot correctly with the new SW image: if this is the case the “restore to factory defaults” should happyly solve the issue.

In pratice the “restore to factory defaults” can be triggered via two different mechanisms due to the fact that the GUI could not be operational after a reboot with a new SW installed; in particular “restore default factory configutaion ” can be triggered:

1. by GUI Operation Page if the GUI is available
2. by keeping the frontpanel button pushed during a device reboot (for some 5-10 sec)

Due to the fact that a “restore to factory defaults” operation could be required after a SW update of a device, it is warmly suggested to save the device configuration on the PC before making any SW update: in this way it is possible to have available a good device configuration to be used eventually to setup again the device after the new SW has been installed. Obviously some manual configuration file adjustments could be required in order to adapt the saved configuration file to the new SW release. The SW release notes will try to advice regarding the need of any configuration file adjustment if any is required.

9 Dual-Mode LoRa implementation scenarios

One of the new features supported by SW vr. 5.x is the dual radio supported by the latest Sarimesh LoRa Tracker HW devices.

The dual radio architecture allows to have two different second generation LoRa HW modules installed and operating simultaneously using either two different antennas or a single one via a suitable power combiner/splitter.

In order to operate this configuration without destroing the LoRa modules and without implementing very complex circuitry at RF level the used approach is to take the benefit of the second generation LoRa modules antenna management to let the two modules happyly coexist with just minor impairement of radio operation, if the modes used by the radio are suitably chosen.

In practice the objective of a dual radio circuit is to be able to receive simultaneously on two different LoRa modes, and be able to transmit just using a single trasmitter.

This imply that during the transmission periods both the radio devices must enter a status in which just one radio transmits and both receivers are antenna-shorted for protection.

The net effect of this is that both radios will not be able to listen during the transmission phases... this is actually already the case of a single device circuitry (LoRa is actually an half-duplex protocol), so it is not out of mind to assume that with two radios the same apply to both the LoRa modules in use.

In practice if we choice, as should be the case, the two Lora modes to be used ortogonal and such that the mode of the primary radio is the one with the greater channel speed (and then the lowest spreading factor if same bandwidth is used for both modes), the actual secondary radio mode impairement should be minimal.
In addition if we choise a CR (coding rate) high, we can benefit of the LoRa capabilities to partially recover the possibly shortned LoRa packets on the secondary radio or simply discard the affected packets with may be a limited increase of the lost packets.

BTW the dual radio is at present an experimental feature to be evaluated on the field... so everything using this mode is to be considered an experimental thing subject to further refinements.

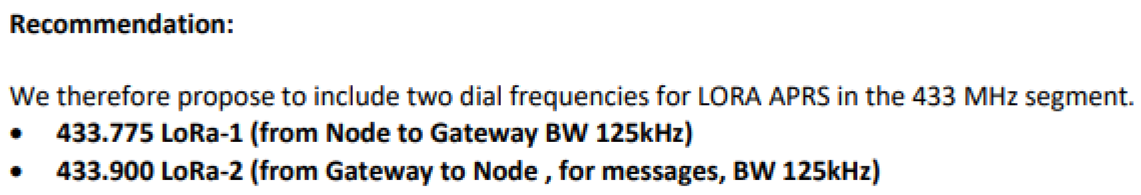
An important topic is how to take benefit of a dual-radio dual-mode device in a LoRa network and how to allow the installed base of legacy devices to benefit of such advanced devices.

Following of this chapter will try to introduce some methods to operate such mixed Lora network configuration scenarios.

First we should clarify why the dual-radio dual-mode has been considered for implementation: at present there is a **IARU recommendation** made few years ago that was proposing a method for experimenting LoRa usage by radio amateurs. The reference document is VIE19-C5-015-OEVSV-LORA-APRS-433-MHz.pdf

This IARU recommendation was based on the LoRa state-of -the art of those years where only first generation LoRa modules were available and not using stable TCXO based frequency control; the actual proposal was the following:

This proposal was then apparently allowing that the LoRa traffic could have, in order to have a bi-


Recommendation:

We therefore propose to include two dial frequencies for LORA APRS in the 433 MHz segment.

- **433.775 LoRa-1 (from Node to Gateway BW 125kHz)**
- **433.900 LoRa-2 (from Gateway to Node , for messages, BW 125kHz)**

Figure 67: IARU Recommendation for LoRa experiments (year 2019)

directional connection, up to 250 KHz of radio band occupancy.

In practice in the past year the dual channel seems never been used and nearly always the LoRa traffic was limited to a single frequency operation using the following LoRa parameters:

BW=125KHz SF=12 CR=4:5

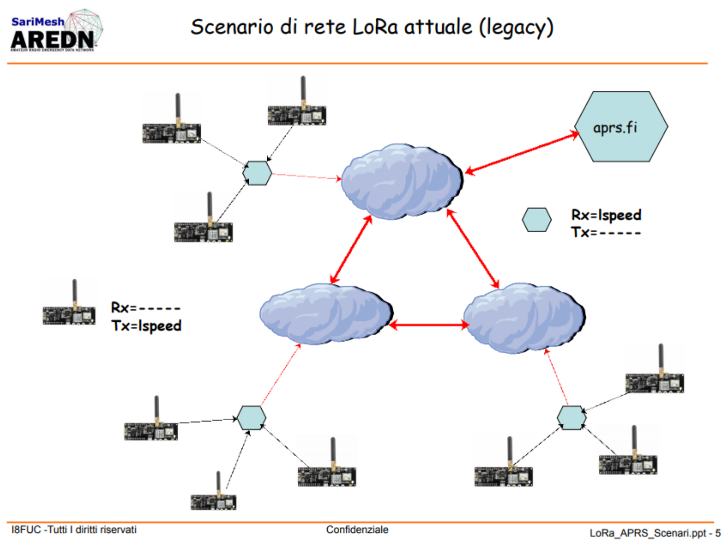

SariMesh
AREDN
Scenario di rete LoRa attuale (legacy)
aprs.fi
Rx=Ispeed
Tx=-----
Rx=-----
Tx=Ispeed
I8FUC -Tutti I diritti riservati
Confidenziale
LoRa_APRS_Scenari.ppt - 5

Figure 68: Present (legacy) LoRa Network scenario

By using these parameters the available radio channel will have a maximum bit rate of 293 bit/sec; if the used CR (Coding Rate) is increased, the available bit rate will still be lower up to 183 bit/sec; assuming a payload length of 52 bytes and a preamble length of 10 symbols we will get an on-air-time of 2.2 secs.

With this level of performancies it is obviously not possible to implement anything more then a very basic payloads content just limited to position and few additional payload infos, very far from what the standard APRS protocol was providing from years.

This was implying that the LoRa APRS application scenario has been till now limited to a set of iGates always directly connected to internet and using a remote portal like aprs.fi to let display the trackers positions; this obviously requires that a user be equipped with a second terminal (may be mobile also) connected to internet via may be the standard 4G public network, then depending on the 4G radio coverage and availability.

In additon a so limited available bit rate does not allow to send anything more then very few position related data, nearly always disabling the possibility to send the classical weather data or telemetry informations, at the expense of an high channel occupancy and a strong imparement of the channel efficiency due to the very limited CAD functionality available on the first generation LoRa chips.

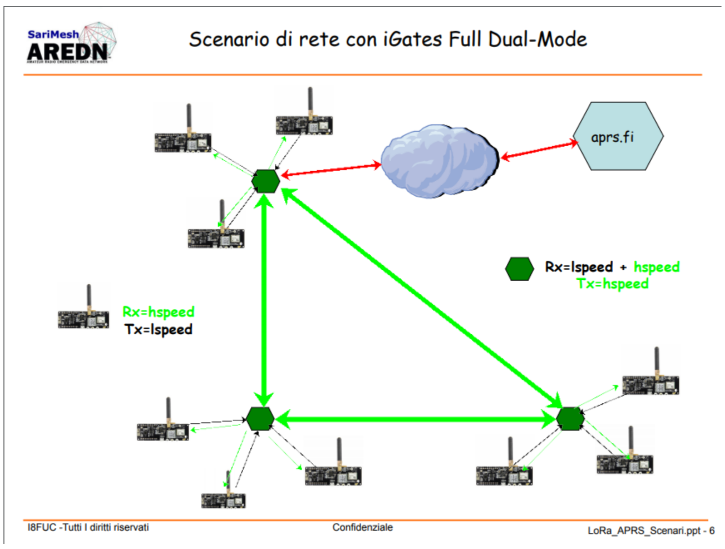
The criteria on which the IARU was formulating its proposal like that, was obviously the LoRa state-of-the-art in those years... but the LoRa technology in the years did a big step forward in introducing new features or refining the original ones, so that now we have much better possibilities to use the radio channel in a more efficient way.

In particular with second generation LoRa modules based on the SX126x chipsets the achivable channel bit rate has been improved by the introduction of lower SF (Spreading Factor) modes and by the introduction of a finally workable CAD (Channel Activity Detection) method (in the first generation chips this was limited only the the packet preamble and thus very inefficient).

The actual availability of these new LoRa modules allows to re-think the usage strategy of the radio channel for our HAM radio applications, taking the benefit in particular of the “**orthogonality**” attribute of the LoRa radio protocol that applies to certain LoRa modes and that allows to use on the same exact radio channel more then a single radio information stream virtually without any reciprocal interference.

The “**orthogonality**” feature of LoRa protocol has been widly deployed in the IoT applications of the LoRa technology, to implement up to 8 different information streams on the same channel independently from the channed size.

There exists also a number of VLSI chips that implement the support of such mode of operation, implementing on the same chip up to 8 different LoRa receivers/decoders; the actual problem with these chips is that they are tigthly designed to operate associated to the IoT LoRaWAN protocol and this make those chips difficult to use by radioamateurs because they cannot use cryptography in their radio equipments.


SariMesh
AREDN
Scenario di rete con iGates Full Dual-Mode
aprs.fi
Rx=lspeed + hspeed
Tx=hspeed
Rx=hspeed
Tx=lspeed
I8FUC -Tutti I diritti riservati
Confidenziale
LoRa_APRS_Scenari.ppt - 6
Figure 69: Full Dual-Mode LoRa Network scenario

Then for HAM radio there is the problem that it is necessary to use a different LoRa receiver for any different LoRa mode to be received. The same problem does not exists for transmitter because in order to avoid very complex circuitry solutions it is necessary to use a single trasmitter per node.

Based on this considerations an alternative proposal for the IARU one that appears easyly achivable is the following:

1) **any iGate node could be equipped with two receivers and a single transmitter operating on different LoRa modes** as follow:

- **legacy mode lspeed (low speed)**: could be the present BW=125 Khz,SF=12,CR=5 in order to guarantie the compatibility with the installed base
- **new hspeed mode (high speed):** could be any LoRa mode orthogonal with the first mode and such that the used values of BW,SF,CR are chosen in order to obtain a channel bit rate well greater then the first legacy mode.

2) **for trackers it will be possible to continue using the single radio devices** just changing the programming so that they would operate in **split mode** (i.e. RX and TX working possibly non different LoRa modes)

The proposed solution allows a smooth migration strategy with minimum impact on the installed base of iGates, due to the fact that it allows to just change the legacy iGate programming in order to operate as **split RX-TX modes (with a single radio)** to be completely workable in the new scenario including dual-radio dual-mode iGates.

Actually the introduction in a legacy LoRa network of dual-radio dual-mode iGates allows to **perform a major quality step up** because it will make available an additional radio channel

(also if just for upstream) binding the iGates at high speed and allowing to **suppress the mandatory internet connection today required** for all the installed iGates.

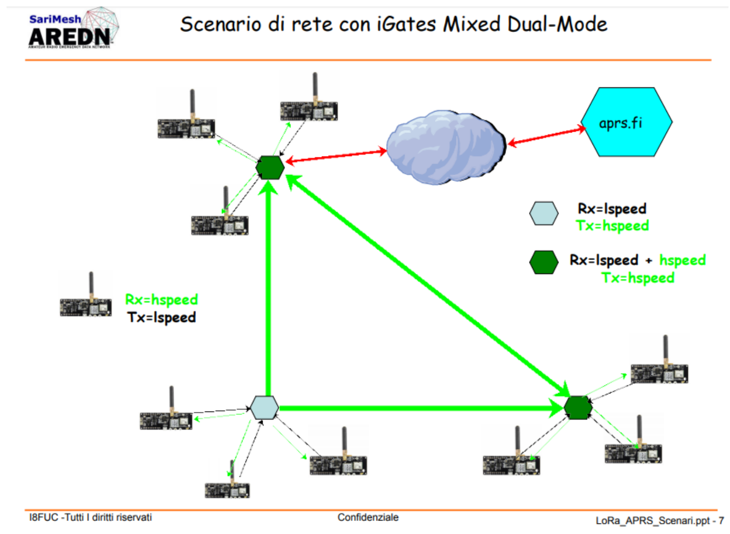

SariMesh
AREDN
Scenario di rete con iGates Mixed Dual-Mode
aprs.fi
Rx=lspeed
Tx=hspeed
Rx=lspeed + hspeed
Tx=hspeed
Rx=hspeed
Tx=lspeed
I8FUC -Tutti I diritti riservati
Confidenziale
LoRa_APRS_Scenari.ppt - 7

Figure 70: Mixed Dual-Mode LoRa Network scenario

The new proposal provides several advantagies with reference the previous status; one of the advantages is that a new iGate can be now installed easly in a remote site also if this site is completely missing internet connectivity, thus lowering the provisioning costs considering that may be the internet connectivity availability is today the major cost for using a remote iGate both for the power consumption and for the internet connectivity cost.

Another big advantage is that by avoiding mandatory internet connection for iGate **it is now possible to implement really off-grid solutions** completely based on the LoRa technology to be usable also in **emergency situations** contrary to what happens with present LoRa scenario.

Obviously now become possible also to program trackers to directly use the hspeed mode thus extending the available bit rate also for trackers and thus enabling the possibility to send more data from trackers, like implementing some sort of LoRa messaging, or send weather station data and telemetry.

All this completely off-grid.

The only drawback is possibly the fact that using lower SF modes the “link budget” at LoRa level will get lower ... this is actually an issue still to be really evaluated in its real impact... infact several studies and user contributions have demonstrated that very often the official link budget made available by Semtech is really meaningless because based on the assumption that the receiver is only affected by the RX input stage noise figure... while it is well known that very often the real limiting factor of a radio link is the actual radio noise level in the receiver site.... often due to the installation of LoRa node close to other radio equipments make it really challenging to have a clean RF spectrum situation on the installation sites.

Anyway it is always possibile in special cases (very clean rx sites) to use the lspeed mode in order to take advantage of the very clean RF status, thus not taking the benefit of increased bit rate.

Fig 68-70 show the different LoRa network scenarios we have described up to now.

10 Debugging Interface

The devices of the Sarimesh LoRa Beacon family are equipped with a remote debug interface that allows access to a series of features designed for advanced use of the devices or to access the functions for monitoring operation in real time without requiring connection to a computer via USB or serial and therefore also usable remotely for devices such as located in a point connected via the Internet.

Access to the Remote Debug interface uses the telnet protocol equipped with a simple password-based security system.

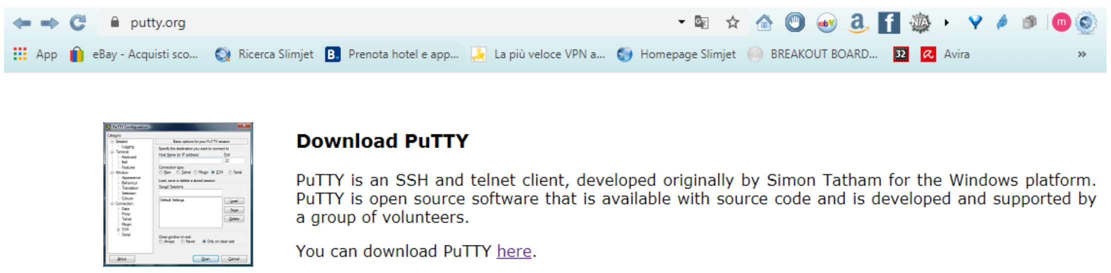
Access to this functionality can take place either using a classic Telnet client available for all existing computer platforms, or via a web application using any internet browser.

Regardless of how you access the Remote Debug interface, a series of commands are available that correspond to as many monitoring or debugging functions.

The two access modes are described below, followed by the debugging commands available.

10.1 Access to Remote Debug IF via Telnet Client

To use this access method, simply obtain any Telnet client such as the classic "putty" downloadable from the following URL: http://putty.org . In fig. 71 you can see the download page.



Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. PuTTY is open source software that is available with source code and is developed and supported by a group of volunteers.

You can download PuTTY here.

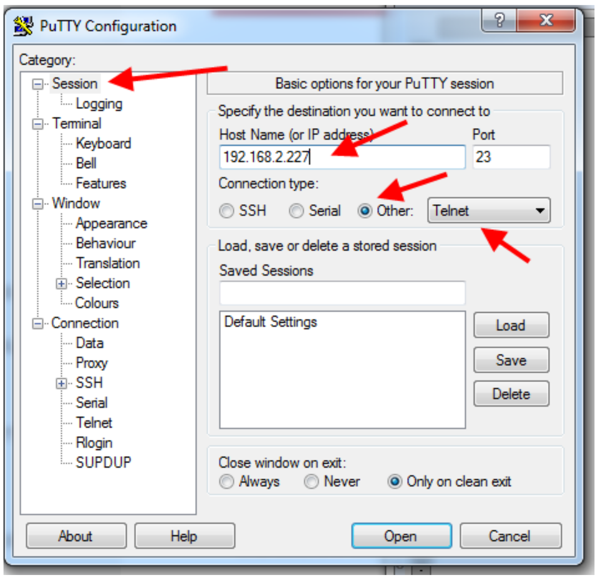
Figure 71: Putty Download page

Once you have downloaded the putty.exe file corresponding to your PC platform that you intend to use, simply start the file without the need for installation.
From the screen of fig.72 , select the following options in addition to the default ones:

- Session: Other --> Telnet
- Host Name: LAN IP address of the device

The device IP could be:
- 192.168.5.1 in case of device in "stand-alone" via WiFi or 192.168.4.1 via Ethernet interface
- a static IP decided by the user in the network 192.168.4.x/24 if using the Ethernet interface
- IP assigned by Wifi router via DHCP if WiFi is used for access.

To discover the device IP it is sufficient to use the front panel button with a double short push and the local display will show it.

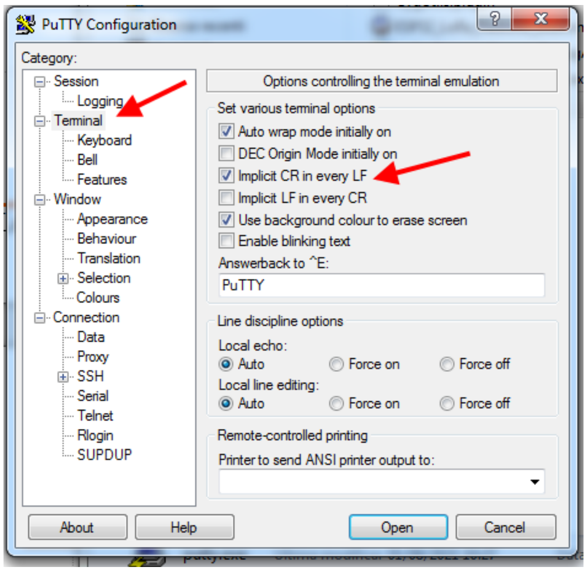


Host Name (or IP address): 192.168.2.227
Port: 23
Connection type: ( ) SSH ( ) Serial [✓ Other:] Telnet
Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

Figure 72: Putty Setup - 1

Follow fig. 73 for the correct settings of Carriage Return and Line Feed:

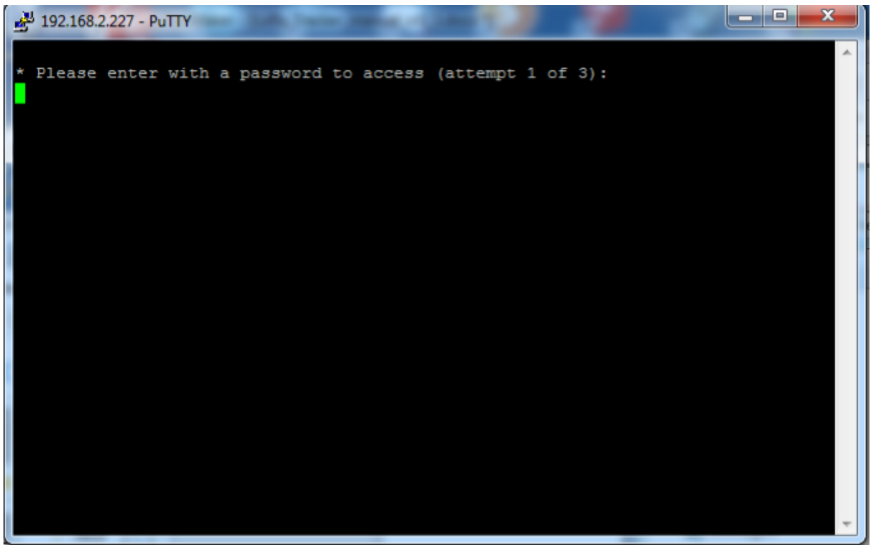
- Terminal: Implicit CR in every LF



[✓] Auto wrap mode initially on
[ ] DEC Origin Mode initially on
[✓] Implicit CR in every LF
[ ] Implicit LF in every CR
[✓] Use background colour to erase screen
[ ] Enable blinking text
Answerback to ^E: PuTTY
Local echo: [✓ Auto] ( ) Force on ( ) Force off
Local line editing: [✓ Auto] ( ) Force on ( ) Force off

Figure 73: Putty Setup - 2

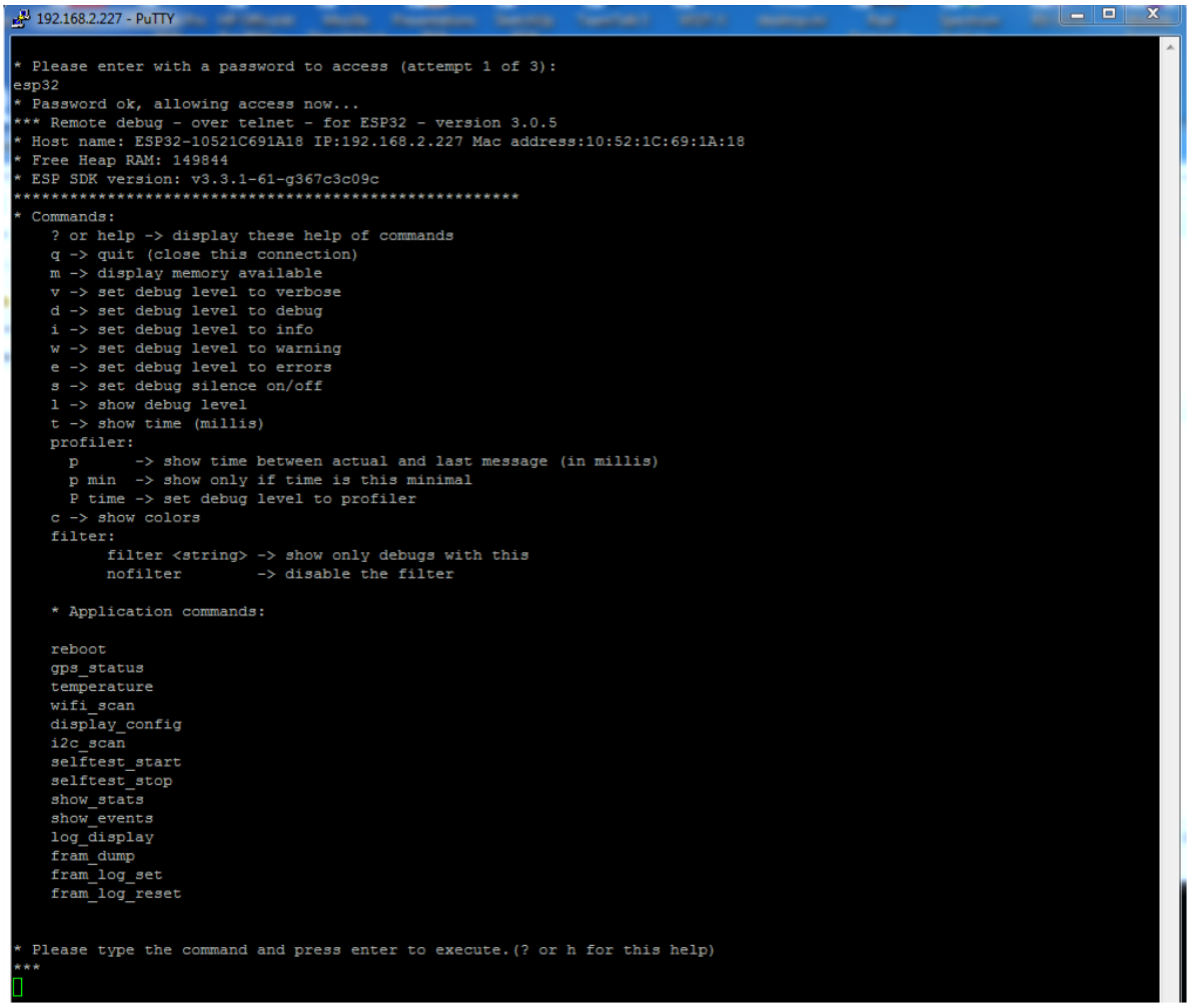
At this point click on “Open”. If the connection is correctly esablished, a screen similar to fig. 74 will appear asking for a password: default is “esp32”.



```
* Please enter with a password to access (attempt 1 of 3):
```

Figure 74 : Initial Telnet screen

By entering the password, the device debug console command interface shown in fig. 75 will appear.



```
* Please enter with a password to access (attempt 1 of 3):
esp32
* Password ok, allowing access now...
*** Remote debug - over telnet - for ESP32 - version 3.0.5
* Host name: ESP32-10521C691A18 IP:192.168.2.227 Mac address:10:52:1C:69:1A:18
* Free Heap RAM: 149844
* ESP SDK version: v3.3.1-61-g367c3c09c
******************************************************
* Commands:
    ? or help -> display these help of commands
    q -> quit (close this connection)
    m -> display memory available
    v -> set debug level to verbose
    d -> set debug level to debug
    i -> set debug level to info
    w -> set debug level to warning
    e -> set debug level to errors
    s -> set debug silence on/off
    l -> show debug level
    t -> show time (millis)
    profiler:
      p      -> show time between actual and last message (in millis)
      p min  -> show only if time is this minimal
      P time -> set debug level to profiler
    c -> show colors
    filter:
          filter <string> -> show only debugs with this
          nofilter        -> disable the filter

    * Application commands:

    reboot
    gps_status
    temperature
    wifi_scan
    display_config
    i2c_scan
    selftest_start
    selftest_stop
    show_stats
    show_events
    log_display
    fram_dump
    fram_log_set
    fram_log_reset


* Please type the command and press enter to execute.(? or h for this help)
***
```

Figure 75: First debug screen

By default, the console reports a series of diagnostic information produced by the device SW, based on the enabled debug functions.

The initial screen, after login, lists the available commands and their meaning briefly; some commands are called "Application Commands" and are used to activate specific functions of the SW.

Some of the available commands represent the exact equivalent of what can be viewed on the serial console via USB typically accessible from within the PlatformIO environment.

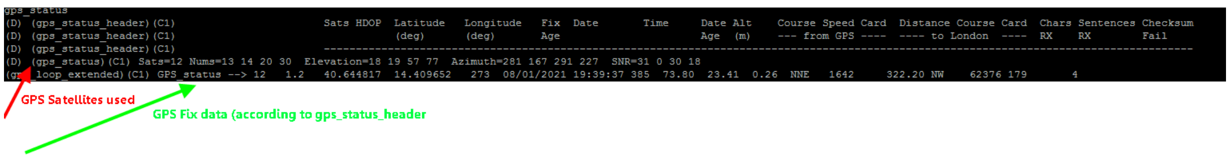
Other commands are additional functions available only through this Remote Debug interface and are described in detail below.

10.1.1 "gps_status" command

The "gps_status" command returns detailed information on the GPS status: in particular, it provides the following parameters:

- List of satellites currently used for the fix: number of satellites used, PRN, elevation, azimuth and SNR values for each satellite used for the fix
- parameters related to the fix: latitude, longitude, age of the fix, date, time, height, speed, quality parameters of the fix

An output example is in fig. 76 .



```
gps_status
(D) (gps_status_header)(C1)                           Sats HDOP  Latitude   Longitude   Fix  Date       Time     Date Alt    Course Speed Card  Distance Course Card  Chars Sentences Checksum
(D) (gps_status_header)(C1)                                      (deg)      (deg)       Age                      Age  (m)    --- from GPS ----  ---- to London  ----  RX    RX        Fail
(D) (gps_status_header)(C1)                           ------------------------------------------------------------------------------------------------------------------------------------
(D) (gps_status)(C1) Sats=12 Nums=13 14 20 30  Elevation=18 19 57 77  Azimuth=281 167 291 227  SNR=31 0 30 18
(gps_loop_extended)(C1) GPS_status --> 12   1.2   40.644817  14.409652   273  08/01/2021 19:39:37 385  73.80  23.41  0.26  NNE   1642     322.20 NW    62376 179       4
```

Figure 76 : output of "gps_status" command

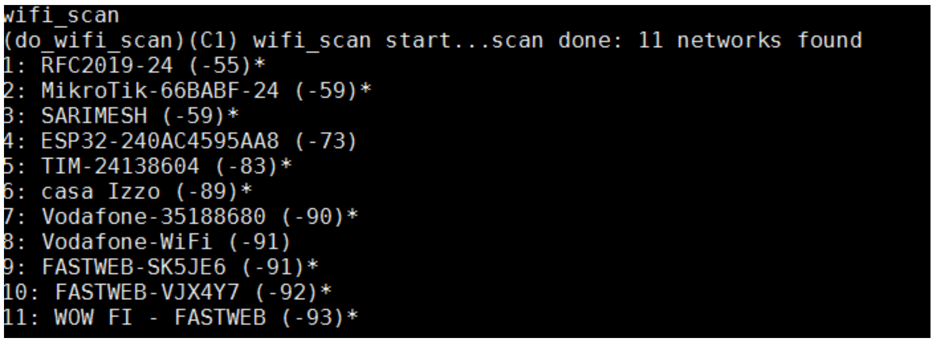
10.1.2 "temperature" command

Provides an estimate of the temperature value in °C at the level of the ESP32 chip inside the device; this value should be used as a general indication of the temperature value as its determination is based on an undocumented feature of the ESP32 chip.

10.1.3 “wifi_scan” command

Scan the 2.4Ghz WiFi band and report the list of networks found and the relative signal levels and the type of encryption.

A typical output is shown in fig. 77 .

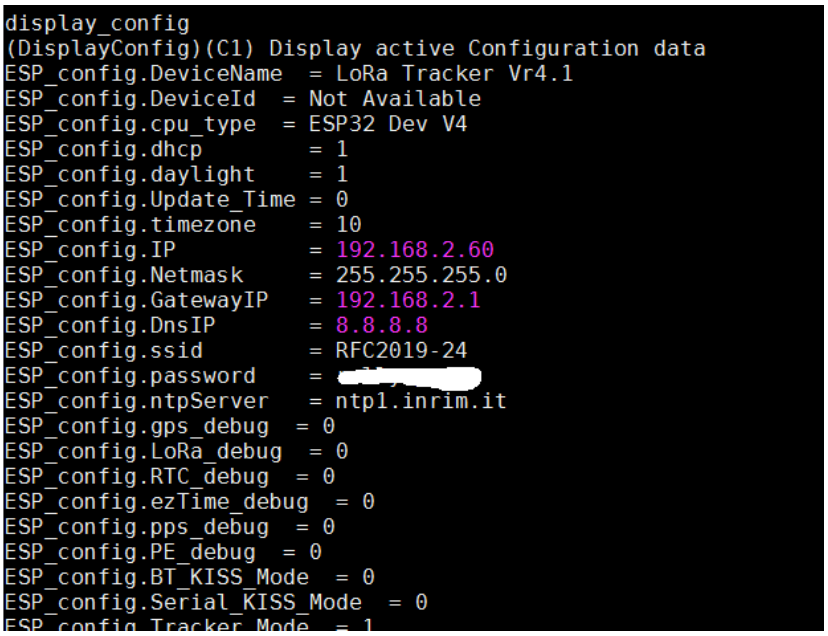


```
wifi_scan
(do_wifi_scan)(C1) wifi_scan start...scan done: 11 networks found
1: RFC2019-24 (-55)*
2: MikroTik-66BABF-24 (-59)*
3: SARIMESH (-59)*
4: ESP32-240AC4595AA8 (-73)
5: TIM-24138604 (-83)*
6: casa Izzo (-89)*
7: Vodafone-35188680 (-90)*
8: Vodafone-WiFi (-91)
9: FASTWEB-SK5JE6 (-91)*
10: FASTWEB-VJX4Y7 (-92)*
11: WOW FI - FASTWEB (-93)*
```

Figure 77 : "wifi_scan" command output

10.1.4 "display_config" command

This command displays in text form the current configuration of the device in use.
The configuration consists of a list of lines of the “key = value” type.
Figure 78 shows only a small subset of parameters for sake of brevity.



```
display_config
(DisplayConfig)(C1) Display active Configuration data
ESP_config.DeviceName  = LoRa Tracker Vr4.1
ESP_config.DeviceId  = Not Available
ESP_config.cpu_type  = ESP32 Dev V4
ESP_config.dhcp        = 1
ESP_config.daylight    = 1
ESP_config.Update_Time = 0
ESP_config.timezone    = 10
ESP_config.IP          = 192.168.2.60
ESP_config.Netmask     = 255.255.255.0
ESP_config.GatewayIP   = 192.168.2.1
ESP_config.DnsIP       = 8.8.8.8
ESP_config.ssid        = RFC2019-24
ESP_config.password    = [redacted]
ESP_config.ntpServer   = ntp1.inrim.it
ESP_config.gps_debug  = 0
ESP_config.LoRa_debug  = 0
ESP_config.RTC_debug  = 0
ESP_config.ezTime_debug  = 0
ESP_config.pps_debug  = 0
ESP_config.PE_debug  = 0
ESP_config.BT_KISS_Mode  = 0
ESP_config.Serial_KISS_Mode  = 0
ESP_config.Tracker_Mode  = 1
```

Figure 78: "display_config" command output

The nomenclature used for the keys should make it easy to identify the meaning of the listed parameters.

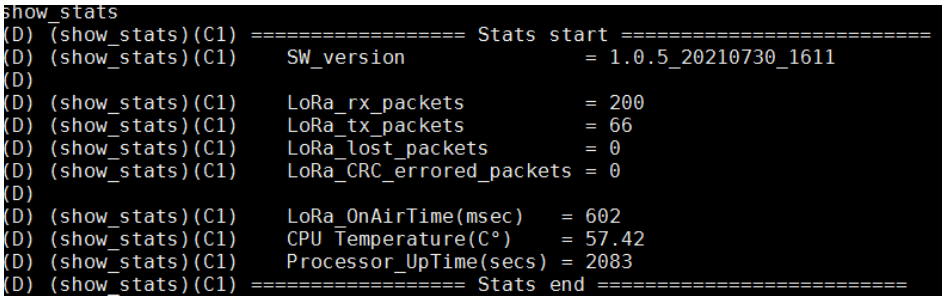
10.1.5 "show_stats" command

This command reports a list of statistics for the LoRa part.

In particular, it reports:

- SW version in use on the device
- Number of LoRa packets received and transmitted
- Number of LoRa packets lost (or suppressed in transmission) due to radio channel congestion
- Number of LoRa packets received with CRC value at the wrong radio level (generally due to transmission errors on the radio channel or access conflict to the radio channel)
- Waiting time for the transmission of LoRa packets
- Temperature of the ESP32 CPU
- "Uptime" time value of the CPU since the last reboot.

Fig. 79 shows an example of this command ouput.



```
show_stats
(D) (show_stats)(C1) ================== Stats start ==========================
(D) (show_stats)(C1)    SW_version               = 1.0.5_20210730_1611
(D)
(D) (show_stats)(C1)    LoRa_rx_packets          = 200
(D) (show_stats)(C1)    LoRa_tx_packets          = 66
(D) (show_stats)(C1)    LoRa_lost_packets        = 0
(D) (show_stats)(C1)    LoRa_CRC_errored_packets = 0
(D)
(D) (show_stats)(C1)    LoRa_OnAirTime(msec)   = 602
(D) (show_stats)(C1)    CPU Temperature(C°)    = 57.42
(D) (show_stats)(C1)    Processor_UpTime(secs) = 2083
(D) (show_stats)(C1) ================== Stats end ==========================
```

Figure 79 : "show_stats" command output

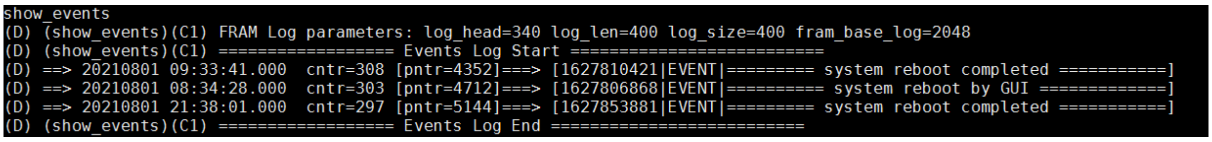
10.1.6 "show_events" command

The device, in the version based on HW LoRa Beacon , has an on-board FRAM memory used for keeping not only the device configuration data, but also a series of data collected in real time related to particular events and to the reception of LoRa spots.

The "show_events" command lists only the "events" part of the log kept in the FRAM; the reported events are typically linked to management actions that imply configuration changes and processor restarts.

The events are "timestamped" with the real time in which they were recorded and therefore they "survive" to device restarts and can be used for troubleshooting even after eventual crash of SW.

Fig. 80 shows an example of output for this command.



```
show_events
(D) (show_events)(C1) FRAM Log parameters: log_head=340 log_len=400 log_size=400 fram_base_log=2048
(D) (show_events)(C1) ================== Events Log Start ==========================
(D) ==> 20210801 09:33:41.000  cntr=308 [pntr=4352]===> [1627810421|EVENT|========= system reboot completed ===========]
(D) ==> 20210801 08:34:28.000  cntr=303 [pntr=4712]===> [1627806868|EVENT|========== system reboot by GUI =============]
(D) ==> 20210801 21:38:01.000  cntr=297 [pntr=5144]===> [1627853881|EVENT|========= system reboot completed ===========]
(D) (show_events)(C1) ================== Events Log End ==========================
```

Figure 80: "show_events" command output

10.1.7 "log_display" command

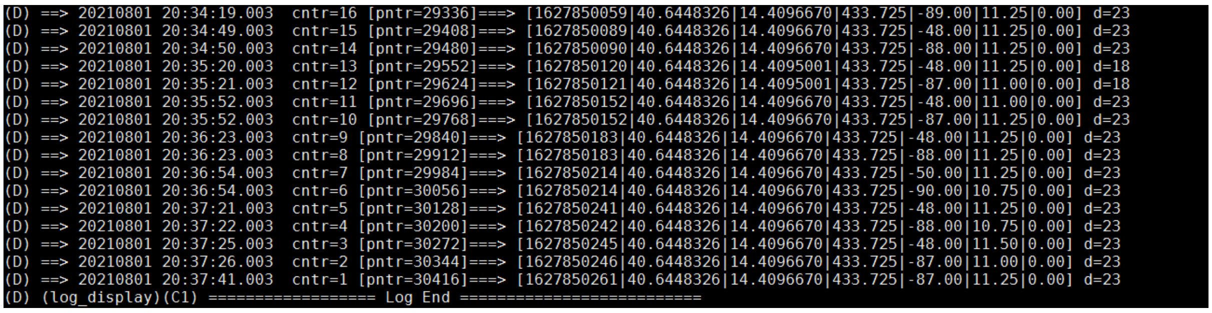
The "log_display" command shows the complete list of the "circular tail" which shows all the events and spots recorded in real time during device operation.

Using a "circular queue" to hold this data allows you to keep track of the last 400 recorded events or spots, discarding the oldest.

The format of the recorded "spots" is different for the case of use as an iGate or as a Tracker:

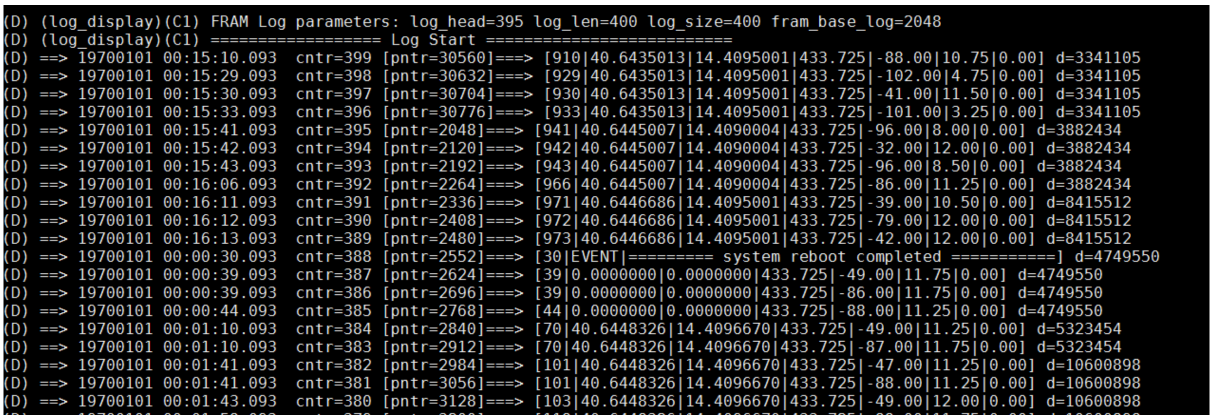
- in the first case (use as iGate) each spot shows the name of the source station, the position, the signal level and SNR of the received spot.

- in the second case (use as a tracker) the spot shows the physical position in which the tracker was at the time of receiving a spot, the frequency, the signal level and SNR of the received spot.



```
(D) ==> 20210801 20:34:19.003  cntr=16 [pntr=29336]===> [1627850059|40.6448326|14.4096670|433.725|-89.00|11.25|0.00] d=23
(D) ==> 20210801 20:34:49.003  cntr=15 [pntr=29408]===> [1627850089|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:34:50.003  cntr=14 [pntr=29480]===> [1627850090|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=23
(D) ==> 20210801 20:35:20.003  cntr=13 [pntr=29552]===> [1627850120|40.6448326|14.4095001|433.725|-48.00|11.25|0.00] d=18
(D) ==> 20210801 20:35:21.003  cntr=12 [pntr=29624]===> [1627850121|40.6448326|14.4095001|433.725|-87.00|11.00|0.00] d=18
(D) ==> 20210801 20:35:52.003  cntr=11 [pntr=29696]===> [1627850152|40.6448326|14.4096670|433.725|-48.00|11.00|0.00] d=23
(D) ==> 20210801 20:35:52.003  cntr=10 [pntr=29768]===> [1627850152|40.6448326|14.4096670|433.725|-87.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:23.003  cntr=9 [pntr=29840]===> [1627850183|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:23.003  cntr=8 [pntr=29912]===> [1627850183|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:54.003  cntr=7 [pntr=29984]===> [1627850214|40.6448326|14.4096670|433.725|-50.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:54.003  cntr=6 [pntr=30056]===> [1627850214|40.6448326|14.4096670|433.725|-90.00|10.75|0.00] d=23
(D) ==> 20210801 20:37:21.003  cntr=5 [pntr=30128]===> [1627850241|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:37:22.003  cntr=4 [pntr=30200]===> [1627850242|40.6448326|14.4096670|433.725|-88.00|10.75|0.00] d=23
(D) ==> 20210801 20:37:25.003  cntr=3 [pntr=30272]===> [1627850245|40.6448326|14.4096670|433.725|-48.00|11.50|0.00] d=23
(D) ==> 20210801 20:37:26.003  cntr=2 [pntr=30344]===> [1627850246|40.6448326|14.4096670|433.725|-87.00|11.00|0.00] d=23
(D) ==> 20210801 20:37:41.003  cntr=1 [pntr=30416]===> [1627850261|40.6448326|14.4096670|433.725|-87.00|11.25|0.00] d=23
(D) (log_display)(C1) ================= Log End ==========================
```

Figure 81 : oldest data of "log_display" output



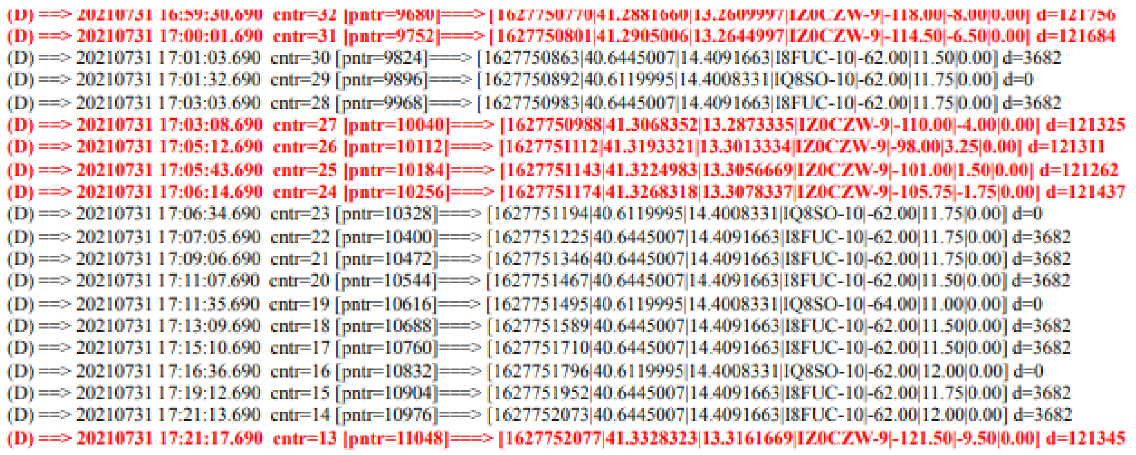
```
(D) (log_display)(C1) FRAM Log parameters: log_head=395 log_len=400 log_size=400 fram_base_log=2048
(D) (log_display)(C1) ================== Log Start ==========================
(D) ==> 19700101 00:15:10.093  cntr=399 [pntr=30560]===> [910|40.6435013|14.4095001|433.725|-88.00|10.75|0.00] d=3341105
(D) ==> 19700101 00:15:29.093  cntr=398 [pntr=30632]===> [929|40.6435013|14.4095001|433.725|-102.00|4.75|0.00] d=3341105
(D) ==> 19700101 00:15:30.093  cntr=397 [pntr=30704]===> [930|40.6435013|14.4095001|433.725|-41.00|11.50|0.00] d=3341105
(D) ==> 19700101 00:15:33.093  cntr=396 [pntr=30776]===> [933|40.6435013|14.4095001|433.725|-101.00|3.25|0.00] d=3341105
(D) ==> 19700101 00:15:41.093  cntr=395 [pntr=2048]===> [941|40.6445007|14.4090004|433.725|-96.00|8.00|0.00] d=3882434
(D) ==> 19700101 00:15:42.093  cntr=394 [pntr=2120]===> [942|40.6445007|14.4090004|433.725|-32.00|12.00|0.00] d=3882434
(D) ==> 19700101 00:15:43.093  cntr=393 [pntr=2192]===> [943|40.6445007|14.4090004|433.725|-96.00|8.50|0.00] d=3882434
(D) ==> 19700101 00:16:06.093  cntr=392 [pntr=2264]===> [966|40.6445007|14.4090004|433.725|-86.00|11.25|0.00] d=3882434
(D) ==> 19700101 00:16:11.093  cntr=391 [pntr=2336]===> [971|40.6446686|14.4095001|433.725|-39.00|10.50|0.00] d=8415512
(D) ==> 19700101 00:16:12.093  cntr=390 [pntr=2408]===> [972|40.6446686|14.4095001|433.725|-79.00|12.00|0.00] d=8415512
(D) ==> 19700101 00:16:13.093  cntr=389 [pntr=2480]===> [973|40.6446686|14.4095001|433.725|-42.00|12.00|0.00] d=8415512
(D) ==> 19700101 00:00:30.093  cntr=388 [pntr=2552]===> [30|EVENT|========= system reboot completed ==========] d=4749550
(D) ==> 19700101 00:00:39.093  cntr=387 [pntr=2624]===> [39|0.0000000|0.0000000|433.725|-49.00|11.75|0.00] d=4749550
(D) ==> 19700101 00:00:39.093  cntr=386 [pntr=2696]===> [39|0.0000000|0.0000000|433.725|-86.00|11.75|0.00] d=4749550
(D) ==> 19700101 00:00:44.093  cntr=385 [pntr=2768]===> [44|0.0000000|0.0000000|433.725|-88.00|11.25|0.00] d=4749550
(D) ==> 19700101 00:01:10.093  cntr=384 [pntr=2840]===> [70|40.6448326|14.4096670|433.725|-49.00|11.25|0.00] d=5323454
(D) ==> 19700101 00:01:10.093  cntr=383 [pntr=2912]===> [70|40.6448326|14.4096670|433.725|-87.00|11.75|0.00] d=5323454
(D) ==> 19700101 00:01:41.093  cntr=382 [pntr=2984]===> [101|40.6448326|14.4096670|433.725|-47.00|11.25|0.00] d=10600898
(D) ==> 19700101 00:01:41.093  cntr=381 [pntr=3056]===> [101|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=10600898
(D) ==> 19700101 00:01:43.093  cntr=380 [pntr=3128]===> [103|40.6448326|14.4096670|433.725|-49.00|12.00|0.00] d=10600898
```

Figure 82: newest data of "log_display" command output

figs. 81 and 82 show a log example for an iGate, while in fig. 83 in case of a tracker.

The reason for the different information content derives from the impossibility, in the tracker case, to identify the source of the LoRa message.

Each spot also contains a field d (distance) whose meaning is left to a future documentation phase; currently it is to be considered a non-significant field.

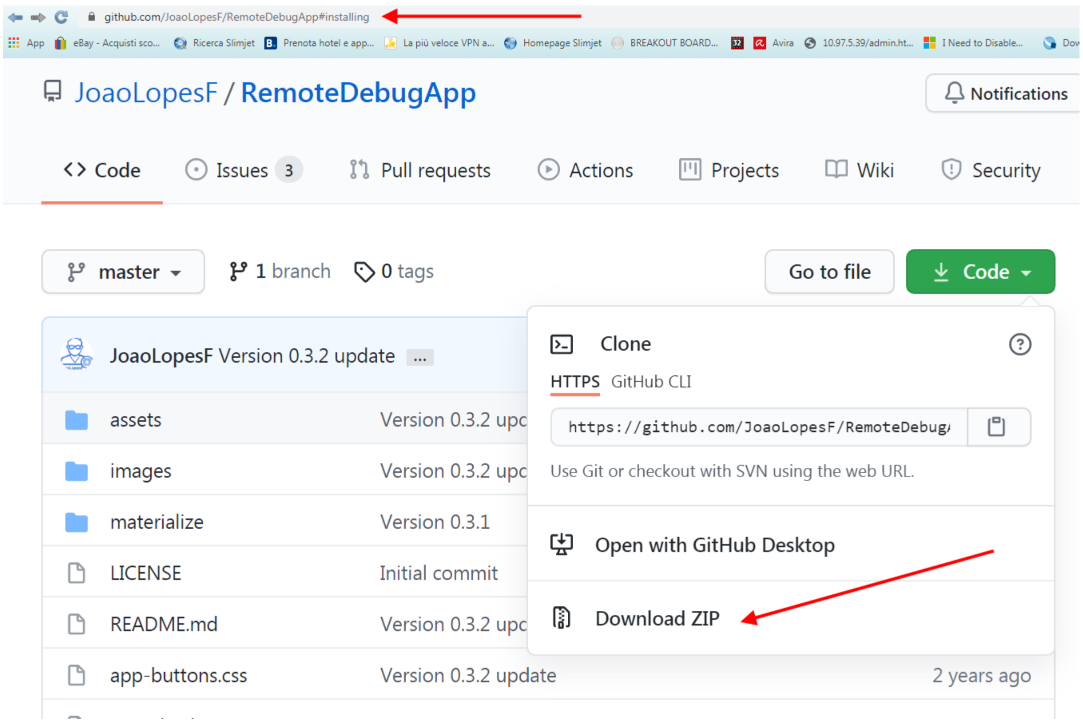


```
(D) ==> 20210731 16:59:30.690  cntr=32 [pntr=9680]===> [1627750770|41.2881660|13.2609997|IZ0CZW-9|-118.00|-8.00|0.00] d=121756
(D) ==> 20210731 17:00:01.690  cntr=31 [pntr=9752]===> [1627750801|41.2905006|13.2644997|IZ0CZW-9|-114.50|-6.50|0.00] d=121684
(D) ==> 20210731 17:01:03.690  cntr=30 [pntr=9824]===> [1627750863|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:01:32.690  cntr=29 [pntr=9896]===> [1627750892|40.6119995|14.4008331|IQ8SO-10|-62.00|11.75|0.00] d=0
(D) ==> 20210731 17:03:03.690  cntr=28 [pntr=9968]===> [1627750983|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:03:08.690  cntr=27 [pntr=10040]===> [1627750988|41.3068352|13.2873335|IZ0CZW-9|-110.00|-4.00|0.00] d=121325
(D) ==> 20210731 17:05:12.690  cntr=26 [pntr=10112]===> [1627751112|41.3193321|13.3013334|IZ0CZW-9|-98.00|3.25|0.00] d=121311
(D) ==> 20210731 17:05:43.690  cntr=25 [pntr=10184]===> [1627751143|41.3224983|13.3056669|IZ0CZW-9|-101.00|1.50|0.00] d=121262
(D) ==> 20210731 17:06:14.690  cntr=24 [pntr=10256]===> [1627751174|41.3268318|13.3078337|IZ0CZW-9|-105.75|-1.75|0.00] d=121437
(D) ==> 20210731 17:06:34.690  cntr=23 [pntr=10328]===> [1627751194|40.6119995|14.4008331|IQ8SO-10|-62.00|11.75|0.00] d=0
(D) ==> 20210731 17:07:05.690  cntr=22 [pntr=10400]===> [1627751225|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:09:06.690  cntr=21 [pntr=10472]===> [1627751346|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:11:07.690  cntr=20 [pntr=10544]===> [1627751467|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:11:35.690  cntr=19 [pntr=10616]===> [1627751495|40.6119995|14.4008331|IQ8SO-10|-64.00|11.00|0.00] d=0
(D) ==> 20210731 17:13:09.690  cntr=18 [pntr=10688]===> [1627751589|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:15:10.690  cntr=17 [pntr=10760]===> [1627751710|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:16:36.690  cntr=16 [pntr=10832]===> [1627751796|40.6119995|14.4008331|IQ8SO-10|-62.00|12.00|0.00] d=0
(D) ==> 20210731 17:19:12.690  cntr=15 [pntr=10904]===> [1627751952|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:21:13.690  cntr=14 [pntr=10976]===> [1627752073|40.6445007|14.4091663|I8FUC-10|-62.00|12.00|0.00] d=3682
(D) ==> 20210731 17:21:17.690  cntr=13 [pntr=11048]===> [1627752077|41.3328323|13.3161669|IZ0CZW-9|-121.50|-9.50|0.00] d=121345
```

Figure 83 : example of log in case of a tracker

10.2 Access to Remote Debug IF via Web App

To use this access method, a particular application called Remote Debug WEB App must be installed on your PC and can be downloaded from the following URL: https://github.com/JoaoLopesF/RemoteDebugApp#installing



```
https://github.com/JoaoLopesF/RemoteDebug/
```

Use Git or checkout with SVN using the web URL.

JoaoLopesF Version 0.3.2 update ...

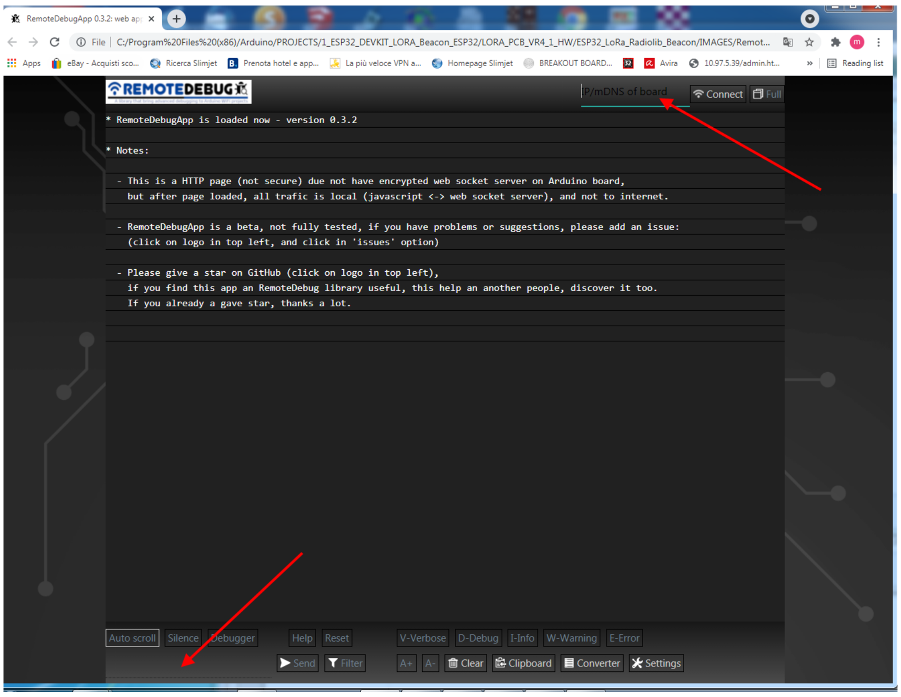
assets	Version 0.3.2 upc	
images	Version 0.3.2 upc	
materialize	Version 0.3.1	
LICENSE	Initial commit	
README.md	Version 0.3.2 upc	
app-buttons.css	Version 0.3.2 update	2 years ago

Figure 84: Remote Debug Web app download page

fig 84 shows the download page; to download the App it is necessary to select the "Code" box and from the subsequent selection window that appears, choose the "Download" option.

The .zip archive thus obtained must be expanded on the PC you are using in a directory chosen by the user; then with the local "explorer" tool, select the index.html file in the directory where the archive was expanded and open it with a browser such as chrome or firefox.

fig. 85 shows the appearance of the resulting chrome browser window.


RemoteDebugApp 0.3.2: web app
C:/Program%20Files%20(x86)/Arduino/PROJECTS/1_ESP32_DEVKIT_LORA_Beacon_ESP32/LORA_PCB_VR4_1_HW/ESP32_LoRa_Radiolib_Beacon/IMAGES/Remot...
REMOTEDEBUG
IP/mDNS of board
Connect
Full
Auto scroll
Silence
Debugger
Help
Reset
V-Verbose
D-Debug
I-Info
W-Warning
E-Error
Send
Filter
A+
A-
Clear
Clipboard
Converter
Settings

```
* RemoteDebugApp is loaded now - version 0.3.2

* Notes:

  - This is a HTTP page (not secure) due not have encrypted web socket server on Arduino board,
    but after page loaded, all trafic is local (javascript <-> web socket server), and not to internet.

  - RemoteDebugApp is a beta, not fully tested, if you have problems or suggestions, please add an issue:
    (click on logo in top left, and click in 'issues' option)

  - Please give a star on GitHub (click on logo in top left),
    if you find this app an RemoteDebug library useful, this help an another people, discover it too.
    If you already a gave star, thanks a lot.
```

Figure 85: Remote Debug first screen appearance

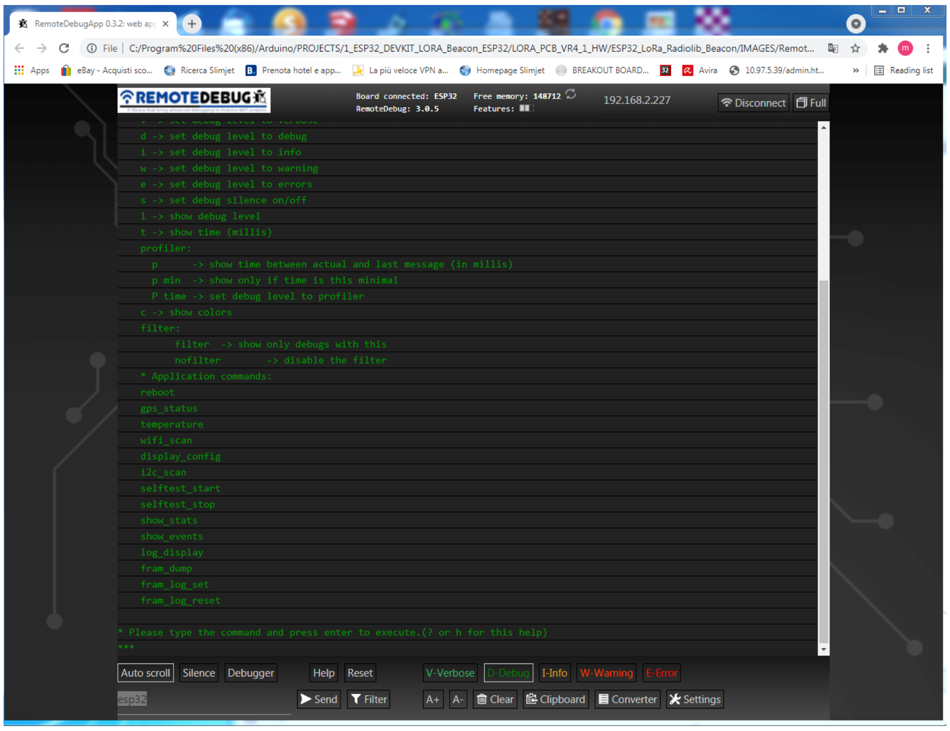
As you can see, it is a classic web interface that initially requires entering the IP address of the device to be monitored.

It is then necessary to enter the access password (by default esp32) in the input window.

After entering the password, you will find a window similar to the one obtained using the putty application described in the previous paragraph.

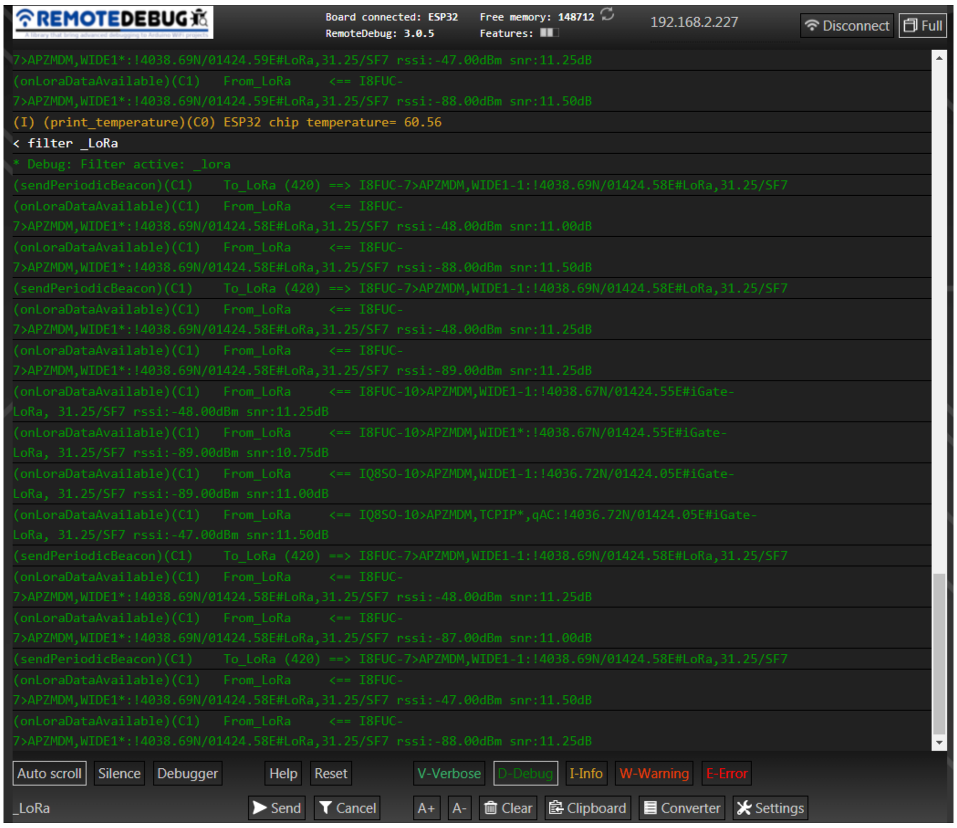
At this point, all the functions and considerations made in the previous paragraph apply exactly.

Using this interface, some additional functions are available, among which perhaps the most interesting is the “Content Filter", used to display only particular messages; this function is therefore very useful in case of different debug functions enabled on the device.
An example of filter application is shown in fig. 86



```
Board connected: ESP32    Free memory: 148712
RemoteDebug: 3.0.5        Features:
192.168.2.227

  v -> set debug level to verbose
  d -> set debug level to debug
  i -> set debug level to info
  w -> set debug level to warning
  e -> set debug level to errors
  s -> set debug silence on/off
  l -> show debug level
  t -> show time (millis)
  profiler:
    p      -> show time between actual and last message (in millis)
    p min  -> show only if time is this minimal
    P time -> set debug level to profiler
  c -> show colors
  filter:
        filter  -> show only debugs with this
        nofilter        -> disable the filter
  * Application commands:
  reboot
  gps_status
  temperature
  wifi_scan
  display_config
  i2c_scan
  selftest_start
  selftest_stop
  show_stats
  show_events
  log_display
  fram_dump
  fram_log_set
  fram_log_reset

* Please type the command and press enter to execute.(? or h for this help)
***
```

Figure 86: Initial Remote debug screen



```
Board connected: ESP32    Free memory: 148712
RemoteDebug: 3.0.5        Features:
192.168.2.227

7>APZMDM,WIDE1*:!4038.69N/01424.59E#LoRa,31.25/SF7 rssi:-47.00dBm snr:11.25dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.59E#LoRa,31.25/SF7 rssi:-88.00dBm snr:11.50dB
(I) (print_temperature)(C0) ESP32 chip temperature= 60.56
< filter _LoRa
* Debug: Filter active: _lora
(sendPeriodicBeacon)(C1)    To_LoRa (420) ==> I8FUC-7>APZMDM,WIDE1-1:!4038.69N/01424.58E#LoRa,31.25/SF7
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-48.00dBm snr:11.00dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-88.00dBm snr:11.50dB
(sendPeriodicBeacon)(C1)    To_LoRa (420) ==> I8FUC-7>APZMDM,WIDE1-1:!4038.69N/01424.58E#LoRa,31.25/SF7
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-48.00dBm snr:11.25dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-89.00dBm snr:11.25dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-10>APZMDM,WIDE1-1:!4038.67N/01424.55E#iGate-
LoRa, 31.25/SF7 rssi:-48.00dBm snr:11.25dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-10>APZMDM,WIDE1*:!4038.67N/01424.55E#iGate-
LoRa, 31.25/SF7 rssi:-89.00dBm snr:10.75dB
(onLoraDataAvailable)(C1)    From_LoRa      <== IQ8SO-10>APZMDM,WIDE1-1:!4036.72N/01424.05E#iGate-
LoRa, 31.25/SF7 rssi:-89.00dBm snr:11.00dB
(onLoraDataAvailable)(C1)    From_LoRa      <== IQ8SO-10>APZMDM,TCPIP*,qAC:!4036.72N/01424.05E#iGate-
LoRa, 31.25/SF7 rssi:-47.00dBm snr:11.50dB
(sendPeriodicBeacon)(C1)    To_LoRa (420) ==> I8FUC-7>APZMDM,WIDE1-1:!4038.69N/01424.58E#LoRa,31.25/SF7
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-48.00dBm snr:11.25dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-87.00dBm snr:11.00dB
(sendPeriodicBeacon)(C1)    To_LoRa (420) ==> I8FUC-7>APZMDM,WIDE1-1:!4038.69N/01424.58E#LoRa,31.25/SF7
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-47.00dBm snr:11.50dB
(onLoraDataAvailable)(C1)    From_LoRa      <== I8FUC-
7>APZMDM,WIDE1*:!4038.69N/01424.58E#LoRa,31.25/SF7 rssi:-88.00dBm snr:11.25dB
```

Figure 87: Content Filter in Remote Debug app

11 Porting of the LoRa Beacon SW to other HW platforms

The SW of the LoRa Beacon project has been designed to be used not only on the LoRa Beacon HW platform described in the previous sections, but also on similar HW devices which use the same type of microcontroller (ESP32) and which include a set of similar HW functions , such as e.g. a GPS module and/or a LoRa module.

At present, the SW has been ported to two particular cards which represent significant examples of "all-in-one" devices available on the usual Chinese portals.

The "porting" operation consists of the following operations:

- properly configure the PlatformIO for SW development by appropriately selecting the HW platform to support
- analyze the HW structure of the device and appropriately set the SW configuration file "MasterConfig.h".
- carry out a "build" of the SW by connecting directly to the target via USB
- carry out the necessary "non-regression" tests to verify the correct functioning of the HW-SW set, only for the functions that can be supported by the HW in use
- verify that the device functionally behaves as expected.

Once the porting work has been successfully completed, it is possible to generate a complete image of the SW that can be loaded onto the card also using the classic ESP32 microcontroller downlad tool, without using the PlatformIO development environment necessarily.

Building images is very useful for those who intend to use simply the SW without necessarily making changes to it, but using the configuration interface provided for setting up the device and for carrying out any experiments with it.

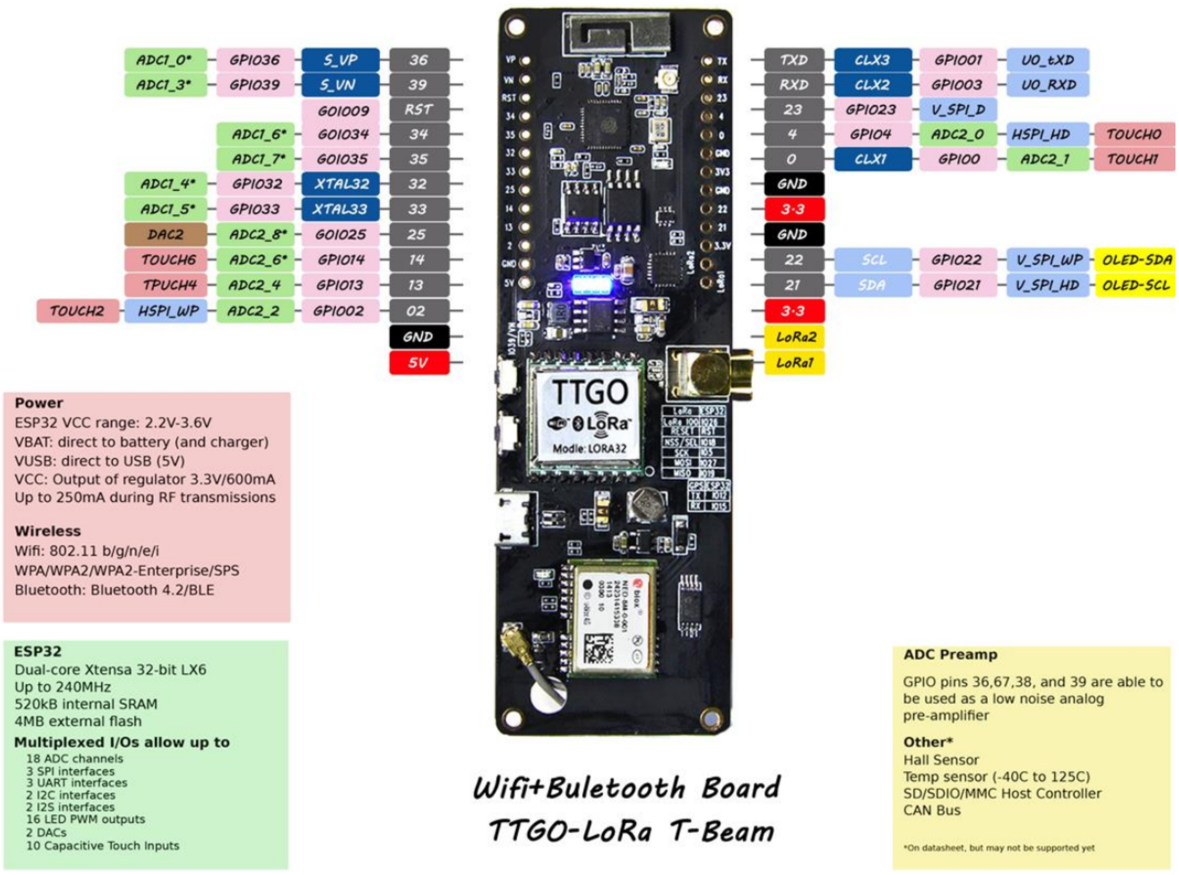
The following paragraphs show the settings to be used for the download tool in relation to the different types of boards to be supported.

11.1 SW installation on TTGO T-beam-V1-2019 device

This card is the first version of a long series of devices made by the same Vendor.

It’s worth to be noted that although these devices share the same commercial name “T-Beam”, they may have a HW equipment that is also quite different from the original version. Doing a search on the internet it is possible to learn more about this topic (https://github.com/Xinyuan-LilyGo/TTGO-LoRa-Series).

The following figure shows the details of the device used for the test.



Power
ESP32 VCC range: 2.2V-3.6V
VBAT: direct to battery (and charger)
VUSB: direct to USB (5V)
VCC: Output of regulator 3.3V/600mA
Up to 250mA during RF transmissions

Wireless
Wifi: 802.11 b/g/n/e/i
WPA/WPA2/WPA2-Enterprise/SPS
Bluetooth: Bluetooth 4.2/BLE

ESP32
Dual-core Xtensa 32-bit LX6
Up to 240MHz
520kB internal SRAM
4MB external flash
Multiplexed I/Os allow up to
18 ADC channels
3 SPI interfaces
3 UART interfaces
2 I2C interfaces
2 I2S interfaces
16 LED PWM outputs
2 DACs
10 Capacitive Touch Inputs

ADC Preamp
GPIO pins 36,67,38, and 39 are able to be used as a low noise analog pre-amplifier

Other*
Hall Sensor
Temp sensor (-40C to 125C)
SD/SDIO/MMC Host Controller
CAN Bus

*On datasheet, but may not be supported yet

Figure 88 : TTGO T-Beam V1 board used for the tests

The following procedure obviously refers to the specific device indicated for obvious reasons of availability of the HW on which to carry out the test.

This board is not able to support the FRAM memory present in the LoRa Beacon project, therefore all the SW functions that require this component are not supported, with the exception of the device configuration part only.

Instead of the FRAMs, a portion of the Flash memory is used to store the device configuration; from a functional point of view there are only minor differences compared to the case of using the FRAM.

As first consequence, the permanent logs and therefore the reconfiguration/reboot events and the log of the received spots are not available on this board.

This card supports only (in the tested version) first generation (sx127x) LoRa chips, therefore with a maximum output power of about 100mWatt, with sensitivity of the radio receiving part not improved and without TCXO support for the stabilization of the emitted frequency.
As for the display, the only supported display is the 0.96” monochrome one with I2C interface.

Fig. 89 illustrates the connections of the display to the circuit board.

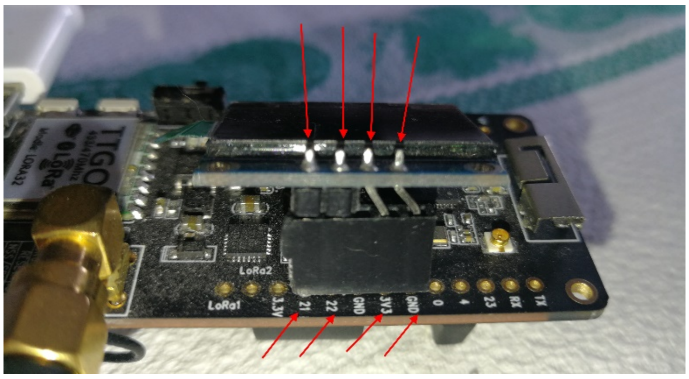
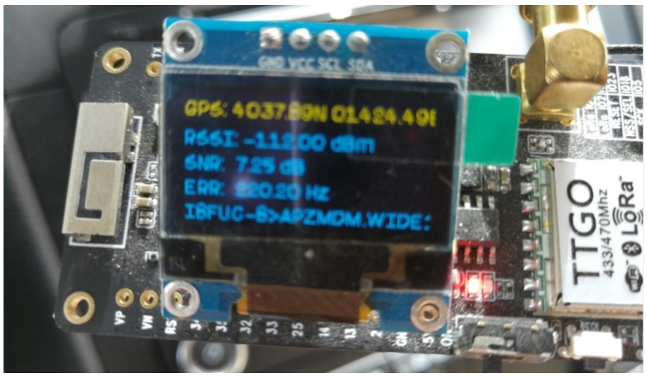


Figure 89 : TTGO T-Beam V1 display connections

The functions of the red and green LEDs are not supported but it is possible to map them to the blue LED present on the devices.

For the SW installation using the download tool presented in the previous chapters, exactly the same considerations apply; in particular also in this case the image will consist in a single file to be loaded on the flash at address 0x0000.

Fig. 90 is an example of the display content.



```
GPS: 4037.89N 01424.49E
RSSI: -112.00 dBm
SNR: 7.25 dB
ERR: 220.20 Hz
IBFUG-8>APZMDM,WIDE:
```

Figure 90 : Example of display content for TTGO T-Beam

Note the presence of the “ **ERR** ” field which reports the frequency error between RT/TX due to the lack of TCXO on this type of HW; this offset can be corrected using the "ppm" field in the setup screen of the LoRa section.

This loading procedure is only required for the first loading of the SW on the card; for subsequent SW updates it is possible to use the OTA update interface available on the GUI.

12 SW installation via OTA (Over-The-Air)

The LoRa Beacon SW provides a way to update the SW without requiring any physical connection to the target device.

This mode, called OTA (Over-The-Air), takes advantage of the WiFi functionality which will be active on the device once the Sarimesh SW is loaded.

The availability of this function allows, moreover, to update the SW of a remote device without needing to be on-site.

The SW update does not impact the configuration data which are therefore preserved between a SW update. However, it remains to be checked after each update because with a new SW image some configuration incompatibiliets can be happening and should be resolbved by adaptiong the old configuration.

Therefore, in order to be able to update the SW, it is necessary to have a single SW module to download onto a computer equipped with a browser of the chrome or firefox family.

The computer to be used for the SW update must be able to reach the device to be updated via WiFi (and possibly via an internet connection): therefore the device to be updated must in turn be connected to an access point from which It is possible to reach (also via internet) the PC that you intend to use for the SW update.

With SW rel. 5.x the OTA SW image loading can also be used locally when the device to be loaded is operating either in the standalone or not standalone mode.

Therefore the first step to be able to carry out the SW update is to verify the connectivity between the PC and the device to be updated, using for example the the classic “ping” tool available on any PC.

Once the connectivity has been verified, it is necessary to place the device to be updated in a particular operating condition called "Admin Mode" which is required in order to temporarily disable some unnecessary functions in the SW update phase.

Note: the LoRa and GPS stages will not work while in Admin Mode!

For this purpose it is possible to access the “OPERATION MODE SETTINGS” screen and select the Admin_Mode box, thus saving the page.

The remote device will now reboot and the display will show “Admin_Mode”.

It should be noted that the automatic reboot is only available using devices based on the LoRa Beacon HW platform; in case of different platforms it will be necessary to manually restart the remote device (and therefore a person near the device or some other mechanism will be needed to restart the device remotely eg by clicking the power).

In these conditions, the device will appear to be temporarily non-functional at the LoRa Radio level as this functionality will, as explained, be temporarily unavailable.

The activation of the Admin_Mode will bring up a new graphic interface dedicated to the SW update: to access this interface it will be necessary to point with another browser window to the address of the remote device, using a port number equal to 88 as in the example of fig. 91

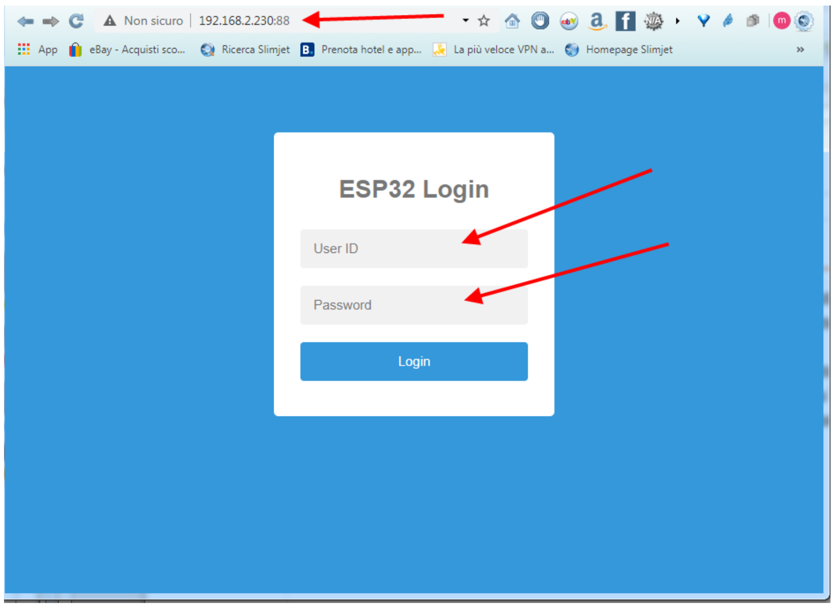


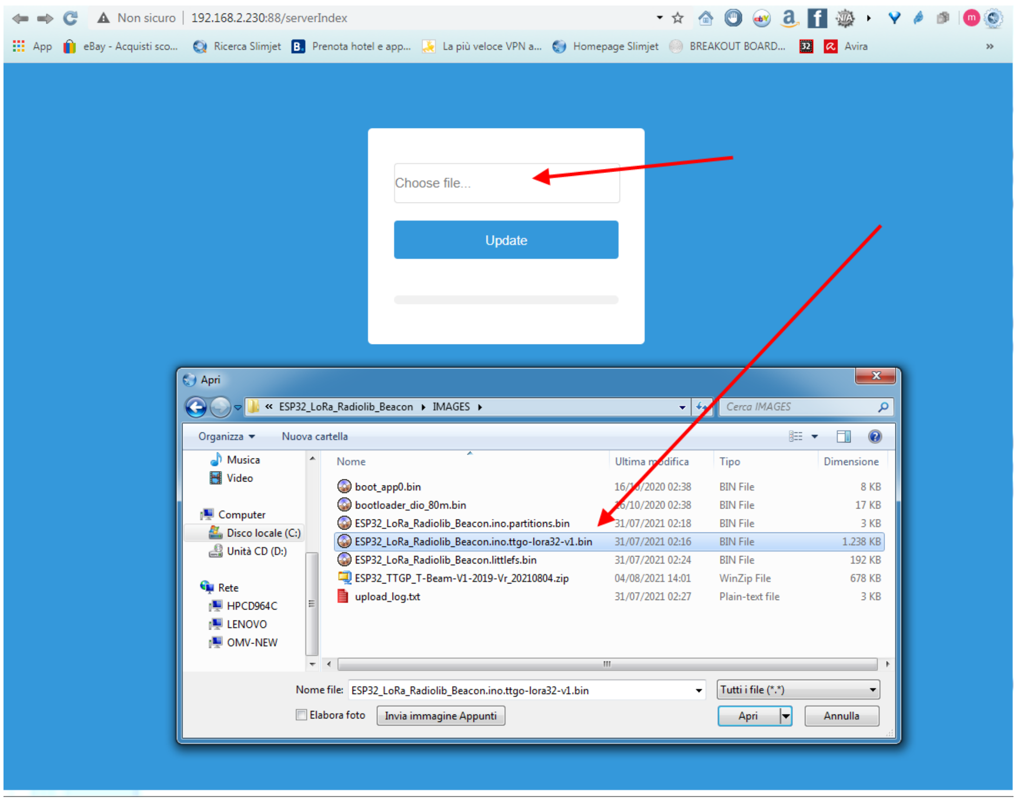
Figure 91: OTA web page

At this point the following values will be entered as credentials (by default):

- User ID: admin
- Password: adminota

A new screen will appear asking you to select the new SW update image from your PC: select the file received as an update, whose name will typically end in .bin.

An example is reported in fig. 92



Non sicuro | 192.168.2.230:88/serverIndex

Choose file...

Update

Apri

« ESP32_LoRa_Radiolib_Beacon ▸ IMAGES ▸

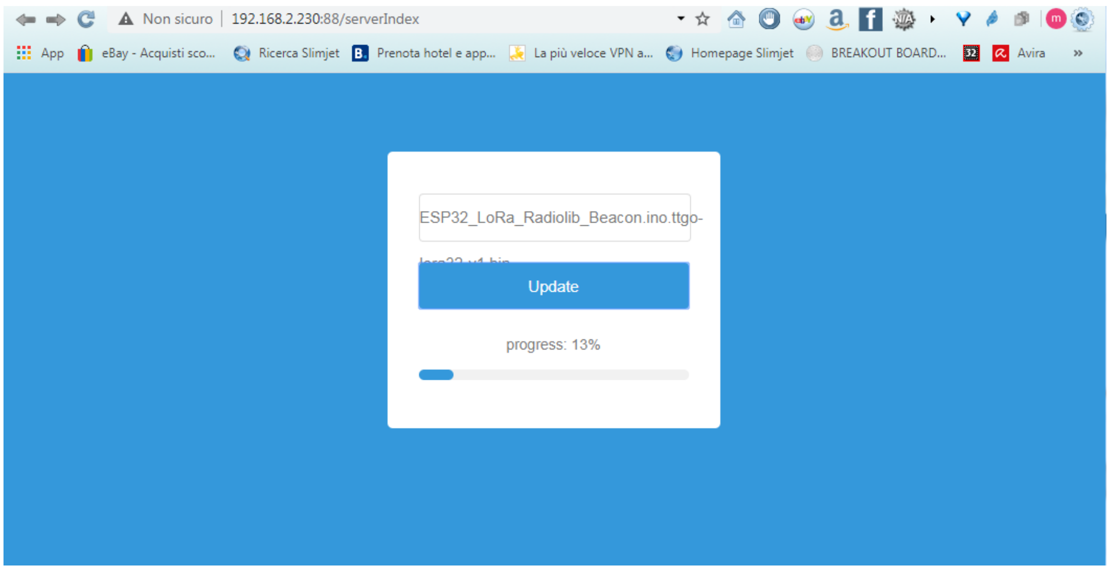
Nome	Ultima modifica	Tipo	Dimensione
boot_app0.bin	16/10/2020 02:38	BIN File	8 KB
bootloader_dio_80m.bin	16/10/2020 02:38	BIN File	17 KB
ESP32_LoRa_Radiolib_Beacon.ino.partitions.bin	31/07/2021 02:18	BIN File	3 KB
ESP32_LoRa_Radiolib_Beacon.ino.ttgo-lora32-v1.bin	31/07/2021 02:16	BIN File	1.238 KB
ESP32_LoRa_Radiolib_Beacon.littlefs.bin	31/07/2021 02:24	BIN File	192 KB
ESP32_TTGP_T-Beam-V1-2019-Vr_20210804.zip	04/08/2021 14:01	WinZip File	678 KB
upload_log.txt	31/07/2021 02:27	Plain-text file	3 KB

Nome file: ESP32_LoRa_Radiolib_Beacon.ino.ttgo-lora32-v1.bin

Tutti i file (*.*)

Figure 92: Selection of SW image to be loaded

Then start loading the SW package and wait for the operation to complete, as reported in fig. 93



Non sicuro | 192.168.2.230:88/serverIndex

ESP32_LoRa_Radiolib_Beacon.ino.ttgo-lora32-v1.bin

Update

progress: 13%

Figure 93 : Progress bar during SW upload

Once the new update image has been loaded, the remote device will reboot and reappear with its graphical interface, according to the new loaded SW; by accessing the "General Configuration" page, it will be possible to verify that the SW version of the device is the one expected as a result of the loading performed.

At this point the device is updated so it is possible to exit the "Admin_Mode" via the "Operation_Mode_settings" page; the device will perform a new reboot and will appear in the selected operating mode.

After each SW update, it is a good idea to check that the configuration of the device is correct as, depending on the changes introduced in the new SW version, minor adjustments to the configuration may be necessary; such adjustments should possibly be documented in the "Release Note" which should accompany each new SW release.

13 Saving and Loading of the device configuration via OTA

The SW LoRa Beacon has been designed to be easily used for experimenting the LoRa radio protocol and not to be a pure SW development exercise, it is aimed as an instrument for the purpose of evaluation the LoRa technology in non-standard uses (according to the objectives for which it was born), or rather for typical use in the amateur radio world.

So the goal was to provide the ability to easily change a series of operating parameters without necessarily having to have SW development experience.

The method chosen was therefore to parameterize all the functional elements and make them modifiable through a graphical interface; the set of parameters that characterize a given setting of each device is indicated hereafter with the term "configuration" and can be viewed as a set of attributes and corresponding values in a standard JSON format.

The configuration of a device is kept in a non-volatile memory which can be a FRAM, in case of LoRa Beacon HW, or a fraction of the flash memory for those devices without FRAM.

The format chosen for the configuration file is the JSON standard which is easily readable and manageable both with a simple editor and with one of the many existing tools for this purpose. These tools can also easily compare multiple configuration files to highlight any differences between them.

The ability to save and reload a certain configuration allows you to easily keep track of the test conditions in which a tests has been performed.

To access the configuration save/restore functions, the “ **SAVE/RESTORE CONFIGURATION** ” page is available.

Fig. 94 shows the content of the “Save/Restore” screen: as you can see, there is a list of config files that can be loaded manually. The File /WebConf.conf always represents the current configuration file.

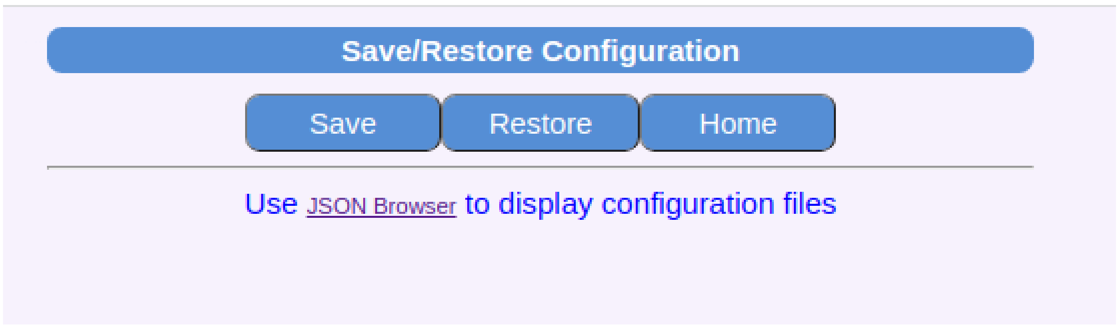

Save/Restore Configuration
Save
Restore
Home
Use JSON Browser to display configuration files

Figure 94 : "Save/Restore" GUI page

To save the current configuration of the device, simply select the “Save” button: a file will be created and downloaded directly to the PC, as shown in the following figure 95:

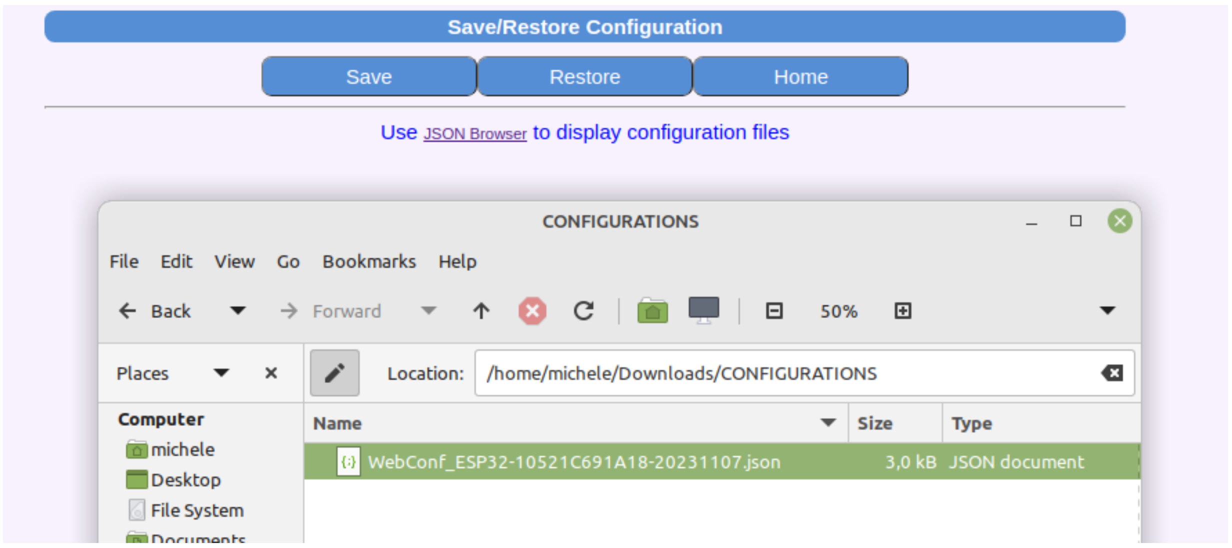


Figure 95 : Saving of the device configuration into a PC

The downloaded file will have a name like “ESP32-<MAC address>.json” and will be saved in a location depending on the browser in use.

To restore the configuration from a file, simply select the "Restore" button and select the file you wish to load, then select the “Upload File” button, as depicted in Fig. 96

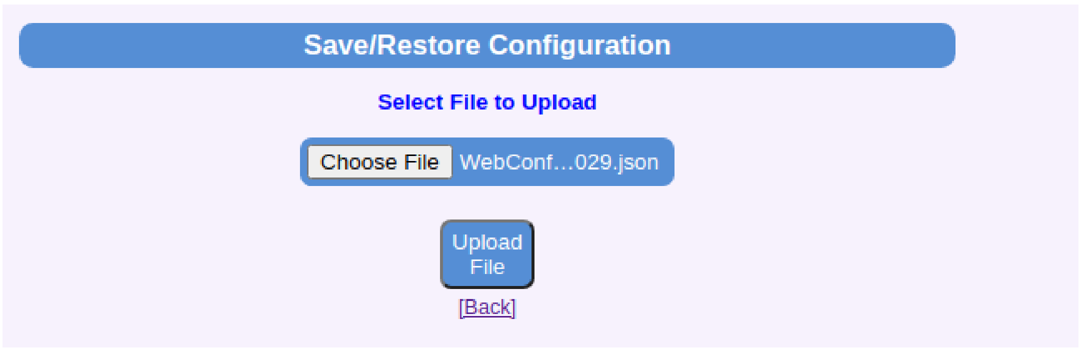
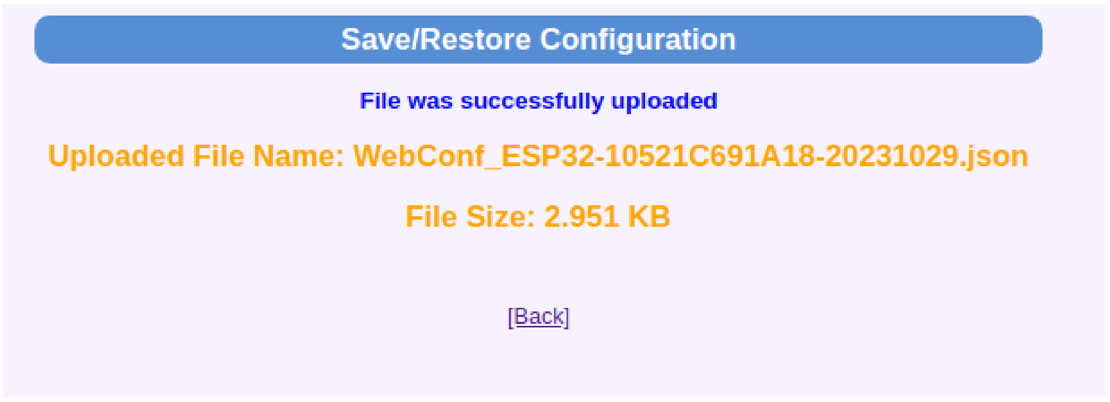


Figure 96 : Restoring configuration from a file



Save/Restore Configuration

File was successfully uploaded

Uploaded File Name: WebConf_ESP32-10521C691A18-20231029.json

File Size: 2.951 KB

[Back]

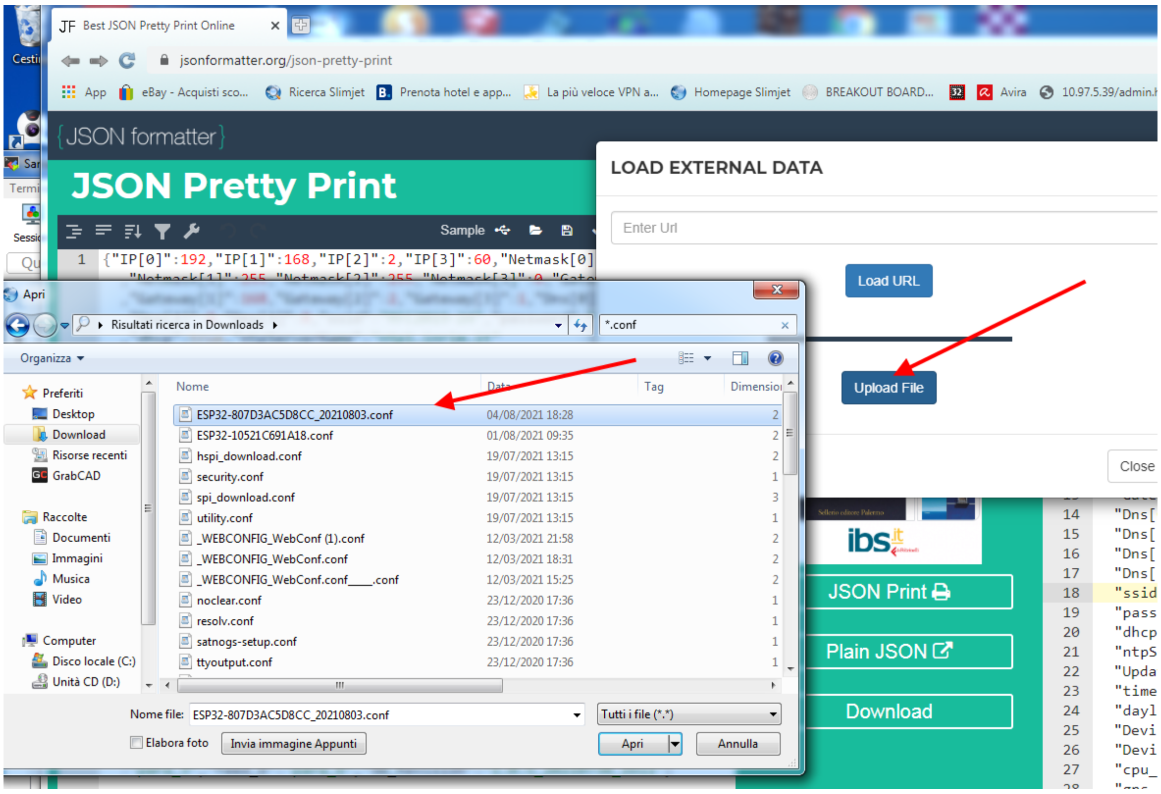
Figure 97 : Successful completion of config restore

If the loading is successfully completed, you will see a screen like fig. 97

To make the new configuration effective, it is necessary to perform a “commit” operation via the Operation and Maintenance GUI page and then reboot the device.

If you want to inspect and/or possibly modify the configuration file, you can use, for example, the tool https://jsonformatter.org/json-pretty-print , which isfreely accessible on the internet via any browser such as chrome or firefox.

In fig 98 all the step for opening a config file are shown.

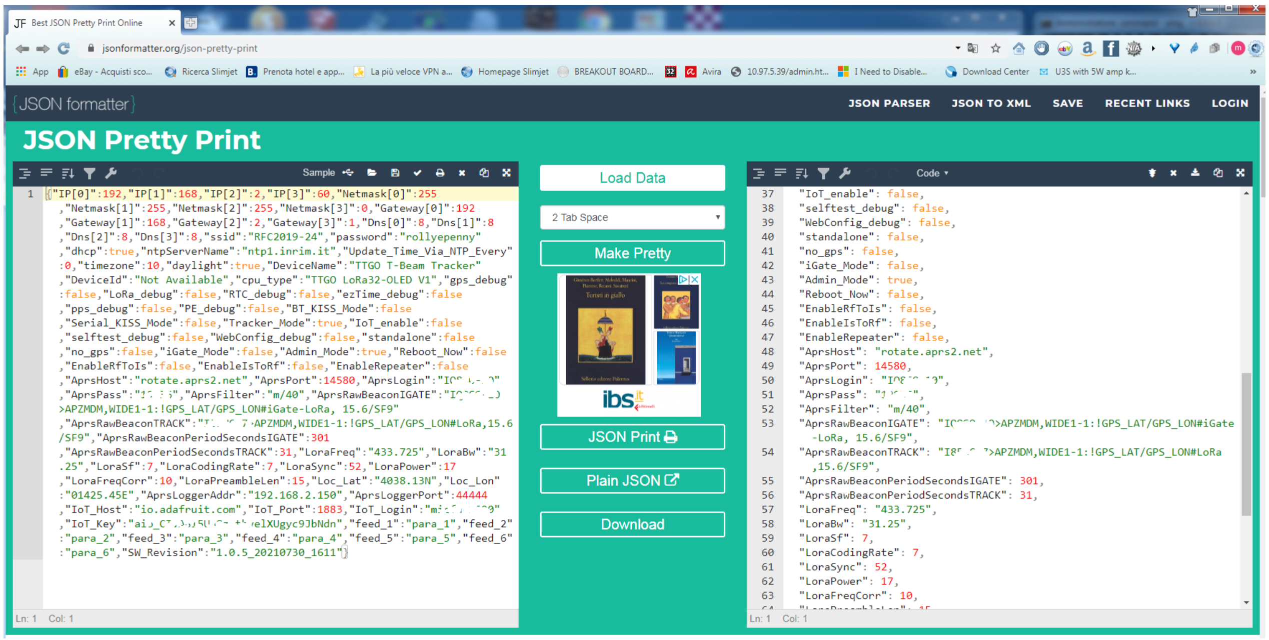


Nome	Data	Tag	Dimensio
ESP32-807D3AC5D8CC_20210803.conf	04/08/2021 18:28		2
ESP32-10521C691A18.conf	01/08/2021 09:35		2
hspi_download.conf	19/07/2021 13:15		2
security.conf	19/07/2021 13:15		1
spi_download.conf	19/07/2021 13:15		3
utility.conf	19/07/2021 13:15		1
_WEBCONFIG_WebConf (1).conf	12/03/2021 21:58		2
_WEBCONFIG_WebConf.conf	12/03/2021 18:31		2
_WEBCONFIG_WebConf.conf____.conf	12/03/2021 15:25		2
noclear.conf	23/12/2020 17:36		1
resolv.conf	23/12/2020 17:36		1
satnogs-setup.conf	23/12/2020 17:36		1
ttyoutput.conf	23/12/2020 17:36		1

Nome file: ESP32-807D3AC5D8CC_20210803.conf
Tutti i file (*.*)

Figure 98 : Opening config file in JSON Pretty Print

The following figure 99 is an example this tool while is viewing the contents of a configuration file in “plain text”:



```
{"IP[0]":192,"IP[1]":168,"IP[2]":2,"IP[3]":60,"Netmask[0]":255
,"Netmask[1]":255,"Netmask[2]":255,"Netmask[3]":0,"Gateway[0]":192
,"Gateway[1]":168,"Gateway[2]":2,"Gateway[3]":1,"Dns[0]":8,"Dns[1]":8
,"Dns[2]":8,"Dns[3]":8,"ssid":"RFC2019-24","password":"rollyepenny"
,"dhcp":true,"ntpServerName":"ntp1.inrim.it","Update_Time_Via_NTP_Every"
:0,"timezone":10,"daylight":true,"DeviceName":"TTGO T-Beam Tracker"
,"DeviceId":"Not Available","cpu_type":"TTGO LoRa32-OLED V1","gps_debug"
:false,"LoRa_debug":false,"RTC_debug":false,"ezTime_debug":false
,"pps_debug":false,"PE_debug":false,"BT_KISS_Mode":false
,"Serial_KISS_Mode":false,"Tracker_Mode":true,"IoT_enable":false
,"selftest_debug":false,"WebConfig_debug":false,"standalone":false
,"no_gps":false,"iGate_Mode":false,"Admin_Mode":true,"Reboot_Now":false
,"EnableRfToIs":false,"EnableIsToRf":false,"EnableRepeater":false
,"AprsHost":"rotate.aprs2.net","AprsPort":14580,"AprsLogin":"IC[redacted]"
,"AprsPass":"1[redacted]","AprsFilter":"m/40","AprsRawBeaconIGATE":"I[redacted]
>APZMDM,WIDE1-1:!GPS_LAT/GPS_LON#iGate-LoRa, 15.6/SF9"
,"AprsRawBeaconTRACK":"I[redacted]>APZMDM,WIDE1-1:!GPS_LAT/GPS_LON#LoRa,15.6
/SF9","AprsRawBeaconPeriodSecondsIGATE":301
,"AprsRawBeaconPeriodSecondsTRACK":31,"LoraFreq":"433.725","LoraBw":"31
.25","LoraSf":7,"LoraCodingRate":7,"LoraSync":52,"LoraPower":17
,"LoraFreqCorr":10,"LoraPreambleLen":15,"Loc_Lat":"4038.13N","Loc_Lon"
:"01425.45E","AprsLoggerAddr":"192.168.2.150","AprsLoggerPort":44444
,"IoT_Host":"io.adafruit.com","IoT_Port":1883,"IoT_Login":"m[redacted]9"
,"IoT_Key":"ai[redacted]velXUgyc9JbNdn","feed_1":"para_1","feed_2"
:"para_2","feed_3":"para_3","feed_4":"para_4","feed_5":"para_5","feed_6"
:"para_6","SW_Revision":"1.0.5_20210730_1611"}
```

Ln: 1 Col: 1

```
  "IoT_enable": false,
  "selftest_debug": false,
  "WebConfig_debug": false,
  "standalone": false,
  "no_gps": false,
  "iGate_Mode": false,
  "Admin_Mode": true,
  "Reboot_Now": false,
  "EnableRfToIs": false,
  "EnableIsToRf": false,
  "EnableRepeater": false,
  "AprsHost": "rotate.aprs2.net",
  "AprsPort": 14580,
  "AprsLogin": "I[redacted]9",
  "AprsPass": "1[redacted]",
  "AprsFilter": "m/40",
  "AprsRawBeaconIGATE": "I[redacted]>APZMDM,WIDE1-1:!GPS_LAT/GPS_LON#iGate
    -LoRa, 15.6/SF9",
  "AprsRawBeaconTRACK": "I[redacted]>APZMDM,WIDE1-1:!GPS_LAT/GPS_LON#LoRa
    ,15.6/SF9",
  "AprsRawBeaconPeriodSecondsIGATE": 301,
  "AprsRawBeaconPeriodSecondsTRACK": 31,
  "LoraFreq": "433.725",
  "LoraBw": "31.25",
  "LoraSf": 7,
  "LoraCodingRate": 7,
  "LoraSync": 52,
  "LoraPower": 17,
  "LoraFreqCorr": 10,
```

Ln: 1 Col: 1

Figure 99 : JSON Pretty Print while displaying config file

14 Suggestions for the components procurement

All the components necessary for the assembly of both versions of LoRa Beacon can be easily purchased on the classic Asian portals; it is important to highlight that although it is generally very easy to find what is necessary, **the experimenter must be careful in purchasing components that are apparently identical to the recommended ones, but much cheaper:** unfortunately it is a classic trap as, like it is well known, in practice there is no guarantee that these cheaper components comply with what is declared or even shown in the photo.

The advice is to look for the deal, but making sure that it is really a deal!

As mentioned, all the components are easily available; the only components obviously not available on these portals are the printed circuits which can be requested at the following address info@sarimesh.net : the price of these PCBs is however very low as it only cover manufacturing costs and shipping.

For year 2023 PCB, due to extensive use of SMD components, we are trying to setup the option to delivery semiassembled PCB already populated with most or all the small SMD components; in this way it should be possible to easyly complete the PCB with only pin-in-hole components to be soldered.

Below is a purely indicative list, at the date, of the main components required for the assembly of the two versions of LoRa Beacon year 2023: the list is unique for the two versions so, based on the version you want to assemble, it will be necessary to purchase only a subset of components. **Of course, no responsibility is assumed for any errors contained in this list, nor for the reliability of the sellers indicated.**

Just in case somebody is interested to this list related to the year 2020 PCB, please check the previous version 4.1 of this document available on the Github or on the Sarimesh site.

Shipping times from China have recently dropped a lot thanks to the new regulation on purchases made on online portals; however, it must be considered that all the prices seen on the portals must then be increased by the value of VAT (22%) if not already included, when the orders are committed.

A special note is to be made regarding the mounting method for the various modules; depending on the fact that the user is making a sample for experiments or for usage only we suggest to use always the mounting via the connectors strips because this makes possible to perform easily any throboul shouting should the mounted device not work immediately ... infact proceding in this way it is easy to remove all the main modules and then insert them one-by-one checking if the different modules are possibly the source of non operation. Please be carafeull to insert new modules while the power supply is down !!!

The following list is updated to 08/11/2023. Only major components are reported; all the SMD parts are not reported and can be anyway collected from the BOM inserted in the description of the various PCB early in this document.

ESP32-WROOM-32D microcontroller 38 pin version with molded antenna

I2C 0.96" display

SPI IPS Display 1.14" 135x240 LCD Module

LoRa module 1 Watt SX1268 Ebyte E22-400M30S
https://it.aliexpress.com/item/1005002748118552.html RA-01S Lora module ipex connector soldered
https://it.aliexpress.com/item/1005005970553639.html T-Display

https://it.aliexpress.com/item/1005001704030454.html GPS
https://it.aliexpress.com/item/1005005530140025.html 3 pin vertical JST XH254 connector
https://it.aliexpress.com/store/910691009 SPS30 sensor
https://it.aliexpress.com/item/4001113450307.html BME680 sensor
https://it.aliexpress.com/item/32849625854.html T-Display heat dissipator
https://it.aliexpress.com/item/4000144606636.html power switch for compact tracker
https://it.aliexpress.com/store/912067512 3mm led assortement
https://it.aliexpress.com/item/1005003826542975.html power splitter
https://it.aliexpress.com/item/33005519676.html LiPo battery
https://it.aliexpress.com/item/1005002450883065.html SMA PCB antenna connector
https://it.aliexpress.com/item/32598945210.html Ethernet module
https://it.aliexpress.com/item/32699746180.html SMA adapter
https://it.aliexpress.com/item/1005002450883065.html SMA connector
https://it.aliexpress.com/item/1005003589507480.html SMA 433 MHz LoRa antenna magnetic
https://it.aliexpress.com/item/1005003155319967.html external GPS antenna
https://it.aliexpress.com/item/1005001386682033.html mini antenna 433MHz conn. SMA

Buzzer KY-012

6x6x12 vertical button
Contact strips 2.54 mm high 7.1 mm female
pin strips 2.54 mm male right angle type R1
DS3231 RTC
IPX-SMA cable 15 cm ipex-SMA pigtail
Straight male 2.54 mm pin strips
12V power connector
https://it.aliexpress.com/item/1005006206380016.html DC/DC steo down conv.

15 APPENDIX 1: LoRa Messenger application

The Sarimesh SW has been structured in a modular way such that it is easy to extend the actual functionalities by addition of new SW modules that could eventually use already existing SW/HW features to implement new top level fucntionalities.

This approach has been already used to implement two major Top Level services; namely the APRS Tracker/iGate functionalities and the Synchronoud Beacon . These services can be enabled/disabled and can eventually coexist according to a suitable resource usage.

In order to allow a possible usage of the HW/SW platform by diverse top level SW modules the SW has been organized so to have some HW resourses usage encapsulated by a suitable "device driver" layer offering a sort of SW API to the external modules; when required the interface between the diffrent SW modules has been organized via a sort of messaging so that the various subssystems can operate in a neraly asynchronous mode, with interaction taking oplace sia a suite IPC (Interprocess Communication method based on messages).

Thanks to this approach it is easy to "port" on our HW/SW environment an existing application with minor changes to it if this application has been structured according a layering approach in its SW architecture.

One of the more interesting funtional additions for our APRS Tracker has beed identified in the addition of an "off-grid" messaging system, to be added to already existing APRS application and able to share the same HW/SW existing modules.

By looking around we selected as a candidate to this addition an existing messaging application developed by Nicholas (https://hackaday.io/project/166225-loramessenger) and available on Github: this application has been very well designed in order to allow an easy porting on other context due to the way the SW has been layered and thanks to not using very language specificites thus allowing a clean SW blocks reuse.

The selected application is a very simple one, but very powerfull and able to run directly over the LoRa level without any intermediate SW layer; the basic working mechanism is described in the original author work and ihas been completely keept in our porting.

It allows any LoRa node to send and receive simple messages (able to fit in a single LoRa PDu (Protocol Data Unit) so that no additional SW layer is required for message size adaptation; any message has a small header wher the destination node or nodes (for broadcast usage) will be indicated; any node that will receive a message will try to understand if that message is just directed to him (and will consume and acknodge that message) or will try to "route" the received message to the destination node according to a "routing" table that the node will keep.
In order to allow the message routing any node will construct in realtime a table where the "hearead" nodes will be enrolled and associated to a quality index derived from the LoRa RSSI of the received message.

In order to allow all node to perform their operation, a beaconing process will take place, by sending time-yo-time suitable "heartbeats" tha will allow the various node to construct their routing tables: this tables will then be update in time so that thies tables will represent for any node the awarness of the actual MoRa messaging network extension.

Operating in this way it is possible that a node will be reachable also if there is not a direct path from the peer node. This is a sort of "meshing" and do not require any dedicated gateway to operate: so it is a perfect “off-grig” communication mechanism for special situations where no suitable LoRa node is present to help different devices to communicate each-other.

The messages received by a generic node can be acknopledged toward the originator node: due to the connectionless communication mechanism there no garatie that these ack will reach the originator nnode: but if a node receives back an ACK this means that the peer is reachable at present.

At the present status the node addressing has been keept at a very minimal level due to the fact that this application is still at an experimental level and is not designed to be scaled to a full messaging system like Meshastin or similar apps. Then at present the node addressing is completely static in the sense that all nodes must be assigned a precise identity in a small range (default is 1-10); in order to corectly operate all nodes must have a unique NodeId.

So at present this application is mostly a proof-of-concept that can be used to eventually extend the actual operation modalities to further complexity.

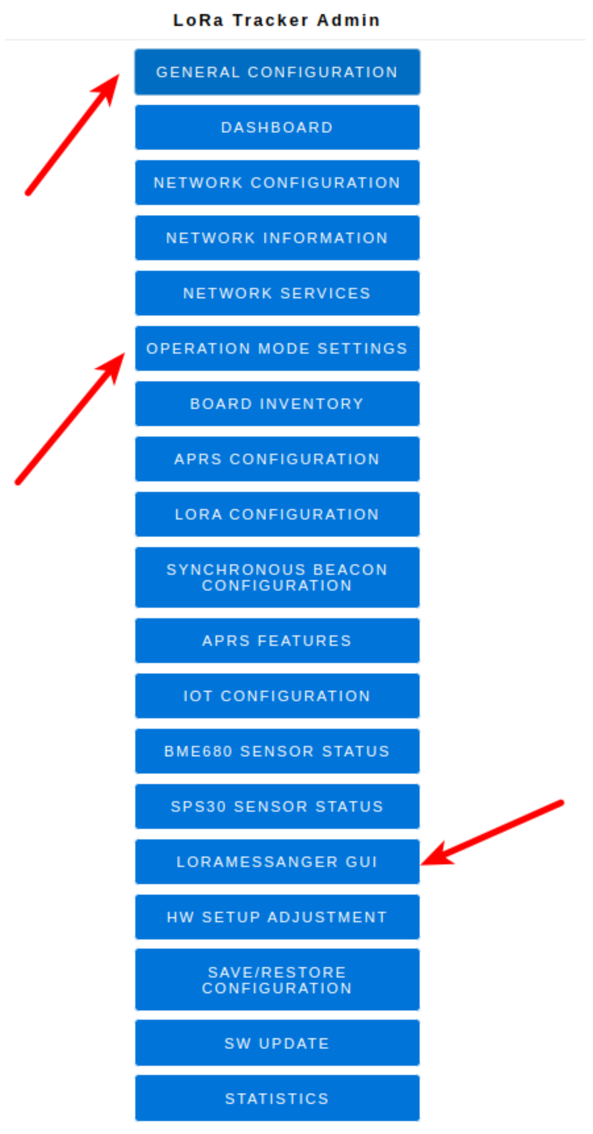


Figure 100- LoRa Messenger Setup

For further details it is possible to check the Nicholas pages over the web.

This application can operate in parallel with the APRS application and puts a light load on the actual radio channel.

To manage this application it needs to be setup by the following procedure, by visiting the GUI pages as in Figure 100

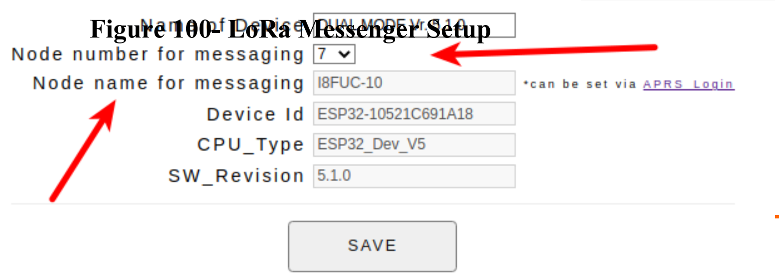
First it will nedd to assign a NodeId to the thaker by visiting the “general Settings” Page of the GUI so that it is possible to associate to any node a numerical ID in a restricted range (see Figure 101).

The actual usage scenario is then that a group of peoples (i.e. a treking group or an emergency group) will set their mobile trackers with known Ids... then it is possible to refer to any node with a mnemonic NodeName derived from the actual APRS SSID of the tracker.

From the specific LM Application screen it is also possible to change the mnemonic name by assigning a full custom name.

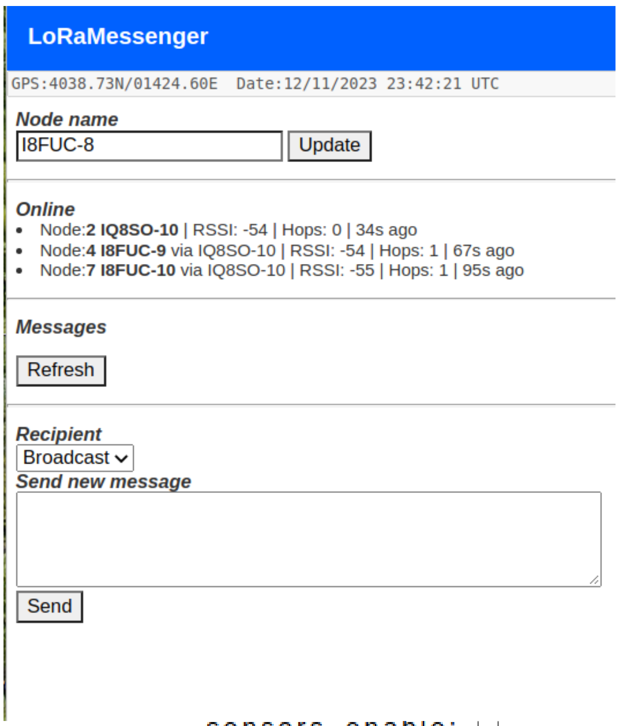
Then the LoRa Messenger application needs to be enabled in the “Operation Mode Settings” GUI page as in Figure 102.

In order to use the LM Application a new GUI page has been created that just allows to read last received messages, write any new message and provide reachable (at LoRa level) active nodes list.



Name of Device	DUAL MODE Vr.5.1.0
Node number for messaging	7
Node name for messaging	I8FUC-10 *can be set via APRS Login
Device Id	ESP32-10521C691A18
CPU_Type	ESP32_Dev_V5
SW_Revision	5.1.0

Figure 101 - LoRa Messenger additional parameters



LoRaMessenger

```
GPS:4038.73N/01424.60E  Date:12/11/2023 23:42:21 UTC
```

Node name

I8FUC-8 [Update]

Online

- Node:**2 IQ8SO-10** | RSSI: -54 | Hops: 0 | 34s ago
- Node:**4 I8FUC-9** via IQ8SO-10 | RSSI: -54 | Hops: 1 | 67s ago
- Node:**7 I8FUC-10** via IQ8SO-10 | RSSI: -55 | Hops: 1 | 95s ago

Messages

[Refresh]

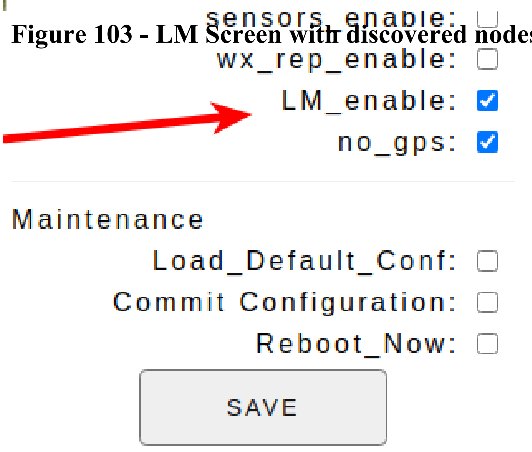
Recipient

Broadcast

Send new message

[Send]

Figure 103 - LM Screen with discovered nodes



sensors_enable: ☐
wx_rep_enable: ☐
LM_enable: ☑
no_gps: ☑

Maintenance

Load_Default_Conf: ☐
Commit Configuration: ☐
Reboot_Now: ☐

SAVE

Figure 102 - LoRa Messenger enable/disable

The actual method to interact with the tracker is via WiFi using any smartphone to access the tracker set to the standalone operation mode (so there is no need to have any external WiFi support because any tracker will present its custom WiFi network like ESP32-<MAC address>.

The connection to the the tracker via the smartphone has been optimized for Android devices (and also IOS..) so that it is sufficient to select the actual WiFi network to connect to and then automagically on the smartphone a web browser window will pop-up with the tracker gui already open.

The smartphone will just operate as a GUI device without contributing in any way to the LM Messaging operation; if any message is received by a tracker, this message will appear as a warning on the tracker display and will trigger the actual user to watch its smartphone if wanted.

Figure 103 is a sample LM GUI screen befor sending any message: actually it referes to a situation where 4 nodes are present in the LoRa reachability area... they appear on the “Online” slice of the screen: for any discovered node there is the Node Id , then the mnemonics associated to that node followed by the value of the RSSI of the received becoan or message , then the Hops count the message was needing to travers (if any) and the time last sign of life was got from that node.

Please note the second and third lines in the Online window: thaey clearly show that the messages fron the nodes was passing trougth an intermediate node.

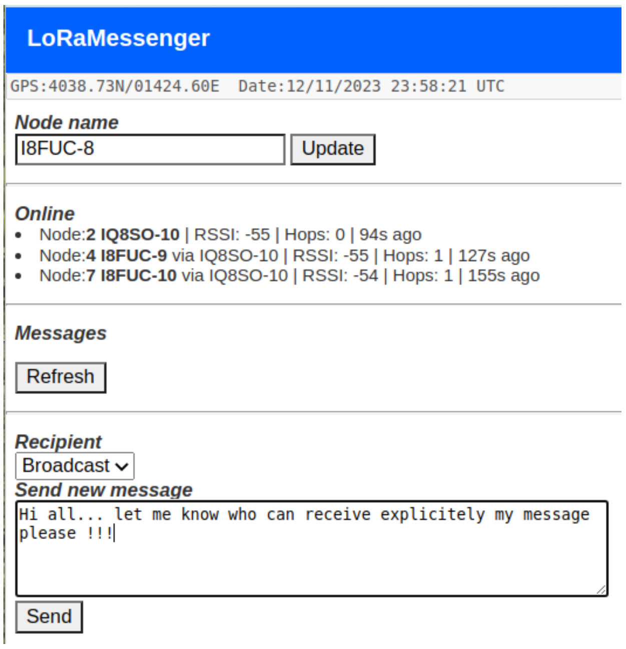
The top line of the screen contains the actual GPS coordinates and the time of day for the traker as derived from the GPS available on the tracker.

The “Node Name” field can be used to set a personal mnemonic identification instead of the default one derived from the APRS tracker name.

The Messages field will report the last messages received by this node and will be refreshed any minute or can be refreshed manually .

The last part of the screen allow to write and send a message: two possibilities are available: either a broadcast to send the message to everybody, or a direct message to a specific node as derived from the above list of Online nodes.

The list of addressable nodes is derived from the online list so that anybody can know the Node Id aand be sure eventually of who is actually associated to a specific mnemonic name.



LoRaMessenger

GPS:4038.73N/01424.60E Date:12/11/2023 23:58:21 UTC

Node name

I8FUC-8 Update

Online

- Node:**2 IQ8SO-10** | RSSI: -55 | Hops: 0 | 94s ago
- Node:**4 I8FUC-9** via IQ8SO-10 | RSSI: -55 | Hops: 1 | 127s ago
- Node:**7 I8FUC-10** via IQ8SO-10 | RSSI: -54 | Hops: 1 | 155s ago

Messages

Refresh

Recipient

Broadcast

Send new message

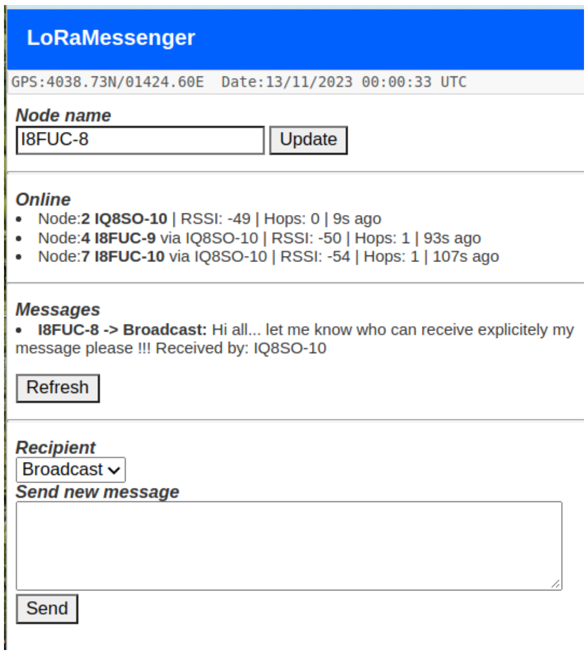
```
Hi all... let me know who can receive explicitely my message
please !!!
```

Send

Figure 104 - Sending a message to broadcast

The figures above show the sending of a message and the reception of a message: if a message will be explicitely acknoledged , this will be reported following the sent message.
If there is not an explicit ack do not mean that the message was not received; infact the present implementation do not guarante that any message acknoledgemnt will be received by the sending node, due to the fact that the ack is also a connection less message.

In the present implementation of the acknoledgemnt mechanisms there are few tricks that should mitigate the “synchronous ack” problem intrinsic in the messaging operation method: infact a sent message will be received by all the nodes in the LoRa reachable area... all these nodes will try to see if the message is to be relayed and just in case will try immediately to perform such operation... In pratice the ack packets will be sent with a randomic delay just in order to try to avoid or mitogate the multiple ack simultaneously... this random delay will add to the CAD mechanism that would in any case operate at LoRa level and will try to delay th simultaneous access to the radio channel by multiple nodes; due to to way used to implement the CAD support in our implementation, a packet that will stay waiting on a positive CAD access grant for more then a defined time, will be discarded, and then the aACK will be lost .



LoRaMessenger

GPS:4038.73N/01424.60E Date:13/11/2023 00:00:33 UTC

Node name

I8FUC-8 Update

Online

- Node:**2 IQ8SO-10** | RSSI: -49 | Hops: 0 | 9s ago
- Node:**4 I8FUC-9** via IQ8SO-10 | RSSI: -50 | Hops: 1 | 93s ago
- Node:**7 I8FUC-10** via IQ8SO-10 | RSSI: -54 | Hops: 1 | 107s ago

Messages

- **I8FUC-8 -> Broadcast:** Hi all... let me know who can receive explicitely my message please !!! Received by: IQ8SO-10

Refresh

Recipient

Broadcast

Send new message

Send

Figure 105 - Received a message...