LoRa Basics™ Modem User Manual

Contents

Introduction

The LoRa Basics™ Modem is an open source software library designed by Semtech. Published as open-source software, it enables communication between Internet of Things devices that use LoRa and Semtech's LoRa Cloud™ services.

It is also designed to run on any host MCU together with an end-device application, making it simple to integrate with a custom platform and use case.

About the different versions of LoRa Basics Modem:

- LoRa Basics Modem version 1.x is for 2.4 GHz communication emulates the LoRaWAN protocol, but uses the 2.4 GHz ISM band to allow worldwide operation and interoperability.

- LoRa Basics Modem version 2.x is only for Sub-G communication based on LoRaWAN protocol, and it supports the LR1110 RF chip. As for the channel plans, in this release it supports CN470 following RP_1B (Regional Parameters version 1B released on 2017).
- The 2.x release doesn't support the LoRa 2.4 GHz protocol with SX128x RF chips. Customers that want to work at 2.4GHz must still refer to the 1.0.1 release.

Though we provide a LoRaWAN stack(lr1mac) in the Basics Modem project, that LoRaWAN stack is not mandatory; users can replace the included stack (lr1mac) with their own private stack.

We also released a single-channel gateway together in the 2.0.0-alpha release for lab tests and demo purposes. Semtech strongly recommends using multichannel GWs for commercial and industrial deployment. This to ensure reliable radio communication and flexible network capacity. More details please check the `Gateway` part.

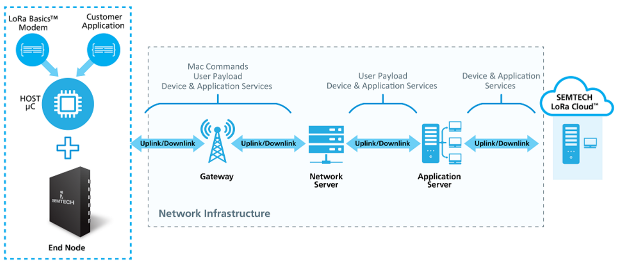


Figure 1: System Architecture

Architecture Overview

The LoRa Edge Modem-E platform supports multiple localization technologies such as GNSS scan and Wi-Fi passive scan. It combines them with a LoRaWAN stack and run into LR1110 SoC, which makes customers easier to develop their own application with less powerful MCU ultimately reducing the cost.

In order to support better other customers who may want to use their own proprietary stack, this LoRa Basics Modem provides a list of similar services to support the same localization technologies in an efficient way, but running into an external host MCU and are released as an open source library.

LoRa Basics™ Modem features two simple-to-use interfaces:

- The Software API that your firmware can call directly for modem configuration, wireless communication and access to high-level LoRa Cloud™ services.
- The Modem Hardware Abstraction Layer that allows the simple adaptation of the Modem to an external MCU.

The modem then takes care of everything else: managing timing to resource planning and allocation, allowing the radio to access the device, and enabling the access to the `Device & Applications Services` provided through Semtech's LoRa Cloud portal. All of this with a fully open-source software architecture intended to facilitate the evolution of the software toward the integration of new features, services and even future MAC protocols.

An efficient multi-protocol-stack feature has been implemented, one that uses a dedicated radio planner. This simplifies radio resource management by preventing direct access to the low-level radio drivers. This innovation allows multiple communication stacks to work concurrently, whilst using the same hardware resources.

The 2.x version of LoRa Basics Modem includes the following services specific for LR1110:

- Application layer Clock synchronization
- Over-the-air (OTA) GNSS almanac update
- OTA GNSS assistance position update
- Wi-Fi passive scan: MAC address extractions
- ROSE streaming code (1 session at the time, 1K Bytes Buffer size, default values supported by LoRa Cloud)

This document explains the API usage of above services demonstrated in the core "Tracker" example, and how to set it up so customers can test it by themselves.

Demo Example

To better guide users to be familiar with this project and how to use it, we provide an demo examples as `user_app/main_examples/main_geolocation.c`.

This demo example is a very simple Tracker. It shows how to:

- Join LoRaWAN network
- Do Almanac update
- Do Wi-Fi Scan
- Do GNSS Passive Scan
- Streaming packets with unknown payload length
- Send packets to NS and obtains by NS based on MQTT
- Access LoRa Cloud DAS Solver to get geolocation results
- Show the geolocation results on AS map

Below is the structure of this application:

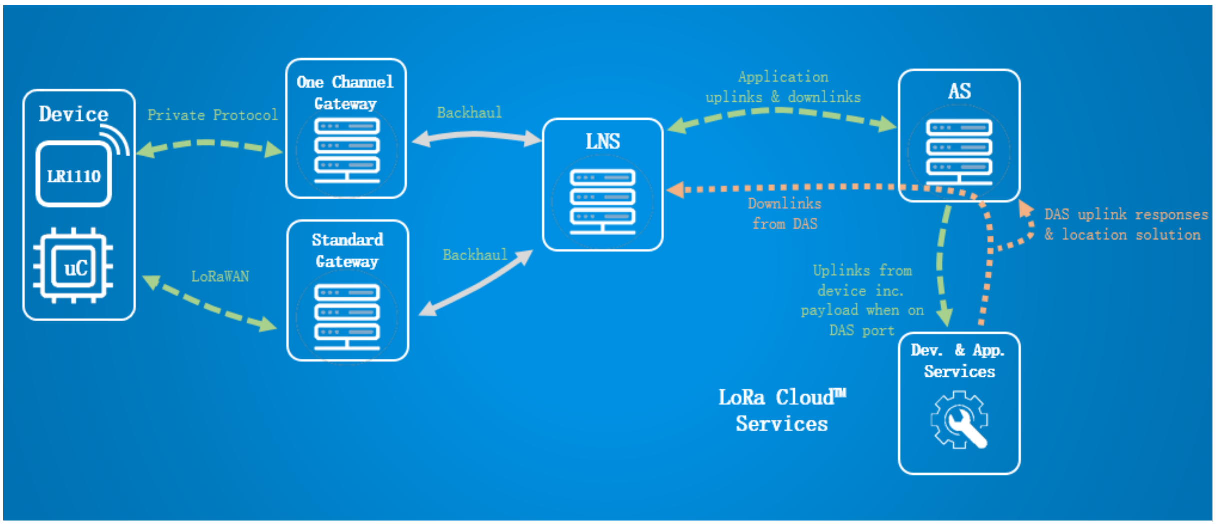


Figure 2: Application Structure

LoRa Basics Modem API

The LoRa Basics Modem source code is released on Github as lorabasicsmodem.

The API head file is `smtc_modem_api/smtc_modem_api.h`. It includes all API functions and is the unique entry point for user who wants to use this LoRa Basics Modem.

The usage of all API functions is in this document: `doc/smtc_modem_api/index.html`.

Hardware

This project supports below hardware for node:

- for version 1.x: STM32L073 + Semtech SX1280 dev kit board
- for version 2.x: Semtech LR1110 EVK, which includes one LR1110 shield combined with STM32L476RG Nucleo-64 board.

Please be noted that the LR1110 shield has two different model:

- `PCB_E592V01B` does not have a LNA and connects to the long antenna
- `PCB_E516V02B` includes one LNA and can only use the short antenna.

These combinations shouldn't be mixed.

Both kind of nodes for version 2.x are showed below:

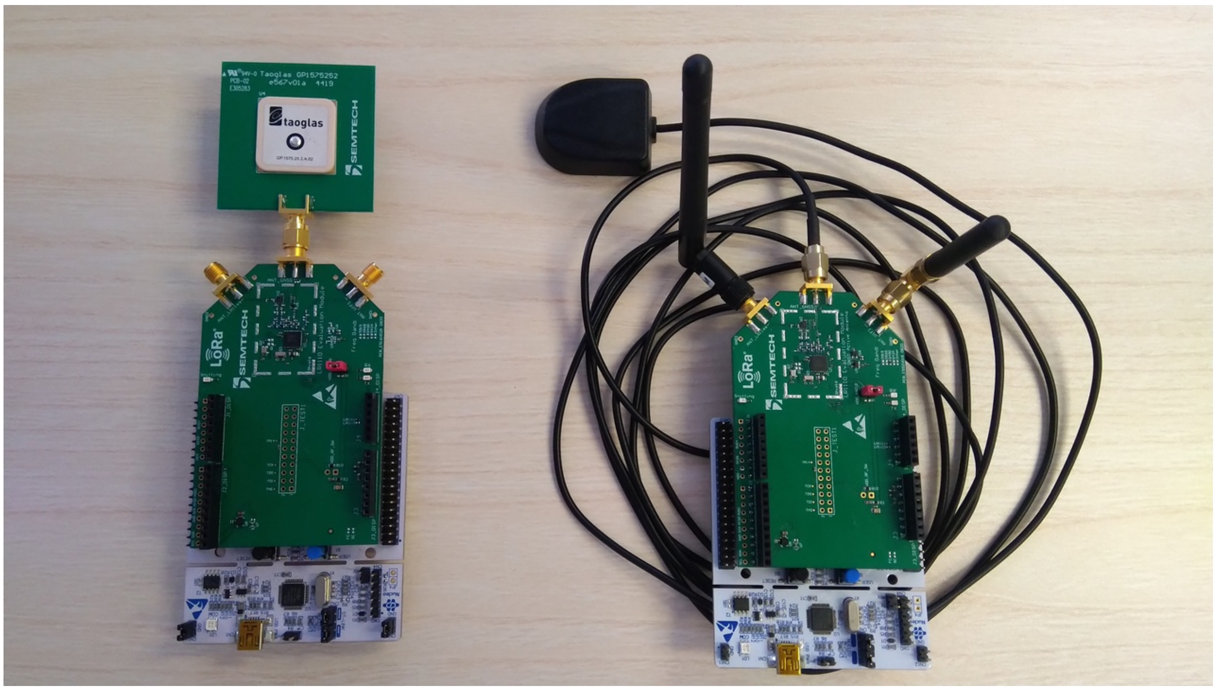


Figure 3: Hardware of Node

Node

This project has an `user_app` folder including an example to show how to make use of the API:

- `main.c`: example entrence; the place to add more examples in the future
- `main_examples/main_geolocation.c`: shows how to perform Wi-Fi scan and GNSS scan, and send the packets out by streaming

The actual functional example is `main_geolocation.c`. It also has a specific support subfolder as `main_examples/geolocation_utilities/`. It is a Tracker application used to demonstrate the usages of some LoRa Basics Modem services.

This Tracker application can periodically do Wi-Fi passive scan or GNSS scan, and send the raw data to some server using the LoRaWAN protocol or customers' own LoRa stack.

This whole demo also includes an AS which will receive such raw data, and inquires Wi-Fi/GNSS Solver to get the geolocation results, and shows the results on a map. You can check the later sections of this document for the details of AS and Solver.

Besides the two kinds of scanning for geolocation function, this demo also shows how to use the other services: `ALC sync`, `Almanac update` and `Streaming`. The first two items are necessary for GNSS scan, while Streaming allows to manage packet with very long payload length without to worry about split it to fragments.

How Tracker works

Since pictures explain things better than words, the picture below shows the timeline starting from system boot to the geolocation position shown on the map.

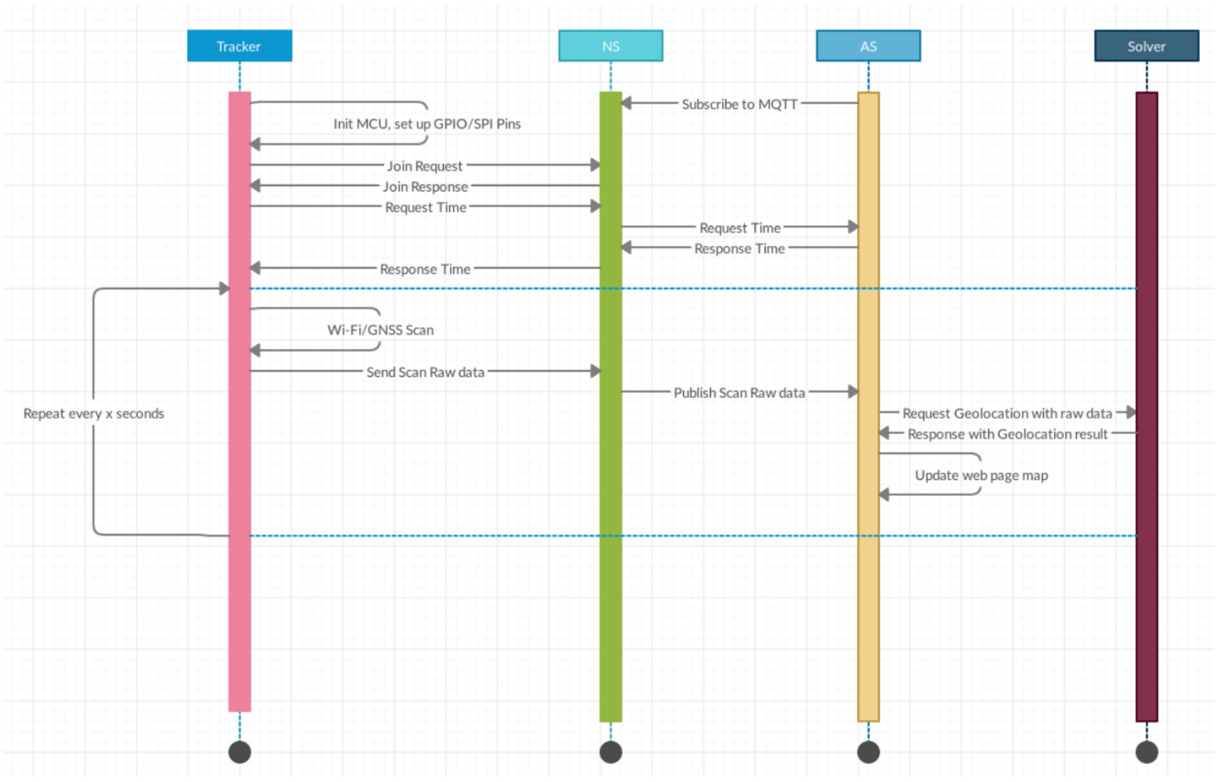


Figure 4: Tracker Wi-Fi Passive Scan Flow

As can be noticed in below picture, there are 4 steps about time updating are only needed for GNSS scan; Wi-Fi Passive scan would start right after the join succeeds.

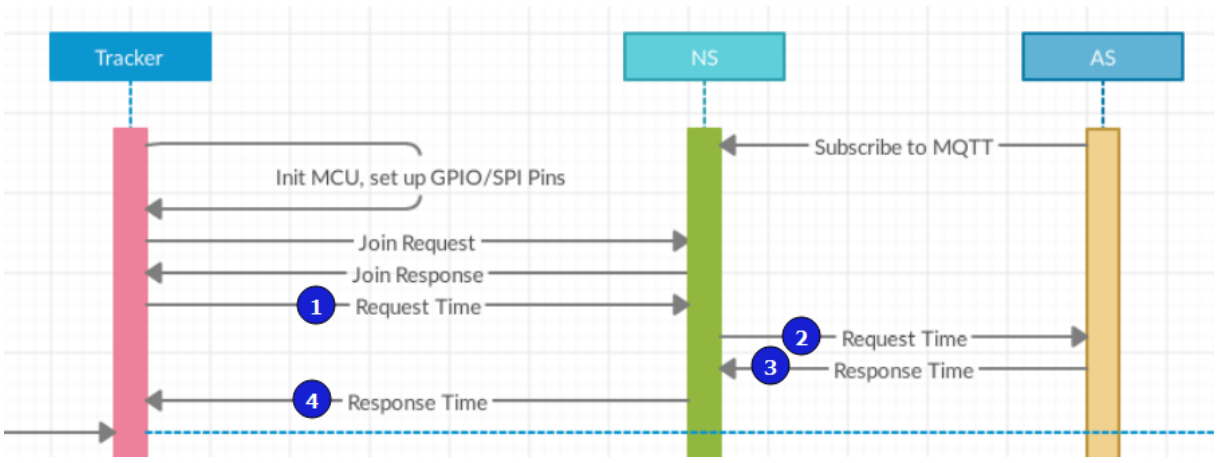


Figure 5: Time update for GNSS Scan

Almanac Update

Almanac update is necessary for GNSS Scan to work correctly, so generally it needs to be done at least once after the LR1110 Firmware upgrade.

The good news is the Tracker has already implemented both manual Almanac update and OTA Almanac update. Depends on customer's requirement, he can use either one or the other.

The tracker source code contains already an Almanac, and the tracker will boot with this almanac. This Almanac will be updated OTA through DAS services if such services are enabled, otherwise the customer must update them manually as described in the `Manual Almanac update` section.

In any case, if DAS services are enabled, OTA almanac update will take place automatically if the Almanac is older than 3 months.

Here is how to implement them:

Manual Almanac update

For manual update, the user needs to have the latest Almanac downloaded from DAS.

In file `main_examples/geolocation_utilities/gnss.h`, there are a few lines of commented Python code. User can copy it out as a simple Python script, replace his/her token (see the section about Resolver for the way to obtain token) for `YOUR_TOKE`, then run it, and redirect the output as a C head file (which would look like `main_examples/geolocation_utilities/almanac_2021_04_06.h`). Then include that C head file in gnss.h replacing `almanac_2021_04_06.h`. After re-compile and flash to Node, that new Almanac data will be used soon after the start of node.

OTA Almanac update

For OTA update, user just needs to call these two functions as below:

```
// Set dm interval to 1 hour for almanac update
smtc_modem_set_dm_info_interval( SMTC_MODEM_DM_INFO_INTERVAL_IN_MINUTE,
    DM_INTERVAL_USER_MIN );

// Active almanac update OTA - WARNING: will remove all other DM
   message
uint8_t info_field = SMTC_MODEM_DM_FIELD_ALMANAC_STATUS;
smtc_modem_set_dm_info_fields( &info_field, 1 );
```

Then the DAS will handle all the OTA Almanac update work as long as the node is connected to the Internet.

Firmware Compile and Upgrade

This project currently only compiles under Linux with ARM GCC.

After installing it, don't forget to add the gcc-arm-non-eabi binary to your PATH environment.

Once ARM-GCC is ready, follow these steps:

- Add your LoRaWAN credentials for joining to NS.

Note: Basics Modem supports **OTAA** (Over-The-Air Activation), but doesn't support **ABP** (Activation By Personalization).

Change below lines in `main_examples/example_options.h` to your own settings:

```
#define USER_LORAWAN_DEVICE_EUI  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00 }
#define USER_LORAWAN_JOIN_EUI    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00 }
#define USER_LORAWAN_APP_KEY     { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, \
                          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
```

- To make GNSS assisted Scan works, you also need to change these lines at the same file:

```
#define MODEM_EXAMPLE_ASSISTANCE_POSITION_LAT   (31.185)
#define MODEM_EXAMPLE_ASSISTANCE_POSITION_LONG  (121.4238)
#define MODEM_EXAMPLE_ASSISTANCE_POSITION_TEXT  "Shanghai, CHINA"
```

- In the Tracker application, both Wi-Fi scan and GNSS scan are enabled by default. If you want to disable one of them, comment one of these two lines accordingly:

```
#define ENABLE_GNSS_SCAN
#define ENABLE_WIFI_SCAN
```

The version 2.x of LoRa Basic Modem supports two kinds of frequency plan for CN470:

- CN470 RP_1B, supports 96 uplink and 48 downlink;
- Mono channel mode, supports only one channel. It is in fact a subset of CN470 RP_1B. This mode can be used for the single channel demo gateway released together for demonstration purpose.

So depends on which type of CN470 frequency plan you want, you have two options when call `make` as:

- `make full`: build to use the whole CN470 RP_1B frequency channels.
- `make full HYBRID_CHINA=yes`: use the single channel mode to work with the single channel demo gateway.

The compiled binary would be `build_modem_lr1110/basic_modem_lr1110.bin`.

Then you can upgrade the firmware now. The whole upgrade includes two phases.

Upgrade LR1110 firmware to LR1110 chip

This phase is optional and can be skipped if the LR1110 shield is already in LR1110 transceiver mode; but users can also do this to upgrade the firmware if they like.

Here is how:

- Download the transceiver bin from `https://github.com/Lora-net/lr1110_updater_tool/wiki`.
- Drag the bin file to STM32L476 temporary USB disk, and waiting for about 20 seconds (or just check the LED on LR1110 shield board; it is done when the Green LED is on).

Upgrade App firmware to STM32L476 board

Follow these steps:

- Drag the earlier compiled `build_modem_lr1110/basic_modem_lr1110.bin` to the STM32L476 temporary USB disk.
- Reset and use the UART terminal to check the output logs with configurations: `Baudrate = 115200; Data bits = 8; Parity = none; Stop bits = 1; Flow control = none`.

Gateway

As stated earlier, this whole LoRa Basics Modem is developed for customers who want to use their own LoRa stack or run LoRaWAN stack in an external MCU, so the demo application can run either on standard LoRaWAN networks and standard LNS (such as the open source NS `chirpstack`), or on some customized LoRa networks. Below we provide a single channel demo gateway as an example of customized LoRa networks.

In the example the demo gateway only supports one channel for transfer, and one channel for receive. In order to simplify the set up, we use 470.9MHz for Tx, and 500.9 for Rx; this way it still works with standard LNS and simplifies the Server side development. But users are free to change with their own gateway and NS as long as the gateway can receive the packets from node, and the NS can work with the AS we provided based on MQTT.

Please be aware of the fact that this single channel example is only aimed to demonstration purposes and not for any implementation in product grade devices.

For more details about this single channel demo gateway as well as how to set it up, please refer to the source code and the document `single-channel-gateway-user-manual.doc`, both located under the subfolder `tools/` under the root directory of LoRa Basics Modem.

Servers for Tracker

Per picture on section `Tracker structure`, there are three different servers involved in this whole system.

- `NS` is Network Server following LoRaWAN protocol or user's private protocol, which is the bridge between node and AS.
- `AS` is Application Server whose main purpose is receiving raw packets from NS, uses those packets to request the geolocation position from Solver, and displays the positions on a map.
- `Solver` is one part of Semtech's LoRa Cloud services, and it will response with the right geolocation position values if the request also provides right TOKENs.

Network Server

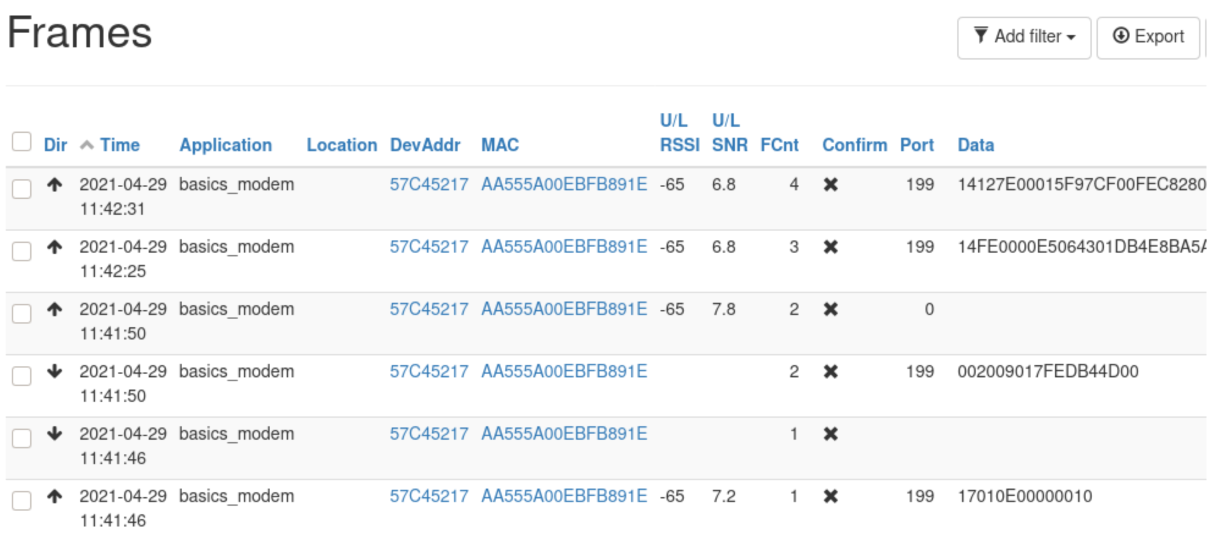
Generally, we need a NS (Network Server) for this Tracker application to work if we use standard LoRaWAN gateway. Otherwise, we can skip this part and connect gateway to AS (although that would require more function added to the implementation of current AS).

As stated earlier, the Tracker can work on either CN RP_1B mode with standard gateways using SX130x, or Mono channel mode with a single channel demo gateway. And it works with generic NS on either mode. So we can use any generic NS available to forward packets.

First, make sure the packets sent from Tracker are received by NS. You may need to register your Gateway to NS first. Check the document for your specific NS about how to do it.

Then, the user needs to create his/her own application or reuse an existed one, and register the node under that application (the node's DevEUI/JoinEUI/AppKey need to be the same as in the application).

Now forward your LoRa packets to your NS by gateway (make sure the IP and port of the NS is rightly set in your gateway), and make sure there are packets received by the NS, such as below picture shows:



☐	Dir	Time	Application	Location	DevAddr	MAC	U/L RSSI	U/L SNR	FCnt	Confirm	Port	Data
☐	↑	2021-04-29 11:42:31	basics_modem		57C45217	AA555A00EBFB891E	-65	6.8	4	✖	199	14127E00015F97CF00FEC8280
☐	↑	2021-04-29 11:42:25	basics_modem		57C45217	AA555A00EBFB891E	-65	6.8	3	✖	199	14FE0000E5064301DB4E8BA5
☐	↑	2021-04-29 11:41:50	basics_modem		57C45217	AA555A00EBFB891E	-65	7.8	2	✖	0	
☐	↓	2021-04-29 11:41:50	basics_modem		57C45217	AA555A00EBFB891E			2	✖	199	002009017FEDB44D00
☐	↓	2021-04-29 11:41:46	basics_modem		57C45217	AA555A00EBFB891E			1	✖		
☐	↑	2021-04-29 11:41:46	basics_modem		57C45217	AA555A00EBFB891E	-65	7.2	1	✖	199	17010E00000010

Figure 6: LoRa Packets received by NS

Solver and Tokens

As explained earlier, the Almanac data can be downloaded from Semtech *Lora Cloud* web site with a right Token. However, user needs to register an account and obtain the Token first.

The whole *LoRa Cloud* includes 3 parts:

- LoRa Cloud Geolocation
- LoRa Cloud Device & Application Services
- LoRa Cloud Device Join

We will use the `Device & Application Services` for the demo and in the following we will refer to it as `DAS`.

Click this *guide* and start from section `Device & Application Services` to learn how to register and obtain the Token.

The *solver address* is rightly set in the AS, so it need not to be changed in the `Application Server` section.

So far, this `solver` is the only part user cannot replace if he needs to use the GNSS geolocation function.

Application Server

We'll simply refer the Application Server as AS, and we provide an example AS for demo based on Node-Red.

Installation

To install the AS, user needs to do 2 steps:

- Install the Node-Red and two libraries
- Import and configure the AS flows released under folder `tools/application_server`

There is a very useful guide dedicated for such work, although is for LoRa Modem-E instead of LoRa Basics Modem server.

Click *this link* to check.

Users are suggested to check it if they are not familiar with Node-Red, or below steps are not clear enough.

Node-Red's default folder is ~/.node-red; but users can specify other folder to replace it. The configuration file is `settings.js` under that folder.

Optionally users can change the username and password with command: `node-red admin hash-pw`, then uncomment this parts in file `settings.js` as below:

```
    adminAuth: {
        type: "credentials",
        users: [{
            username: "admin",
            password: "XXXXXX",
            permissions: "*"
        }]
    },
```

Below are the simple steps to set up the AS.

Install libraries After install Node-Red, we need to install these two libraries:

- `node-red-dashboard`: Provides tools to create dashboards such as a temperature graph
- `node-red-contrib-web-worldmap`: Creates a map for geolocation features

Here is how:

1. Click the menu in the top right corner of the screen.
2. Select `Manage palette`.
3. Select `Install`.
4. Search for `node-red-dashboard`.
5. Click the `Install` button for that package.
6. Click `Install` in the pop-up window.
7. Wait for the installation to complete.

Similarly install the library `node-red-contrib-web-worldmap` follow above steps.

Below pictures should make it clear:

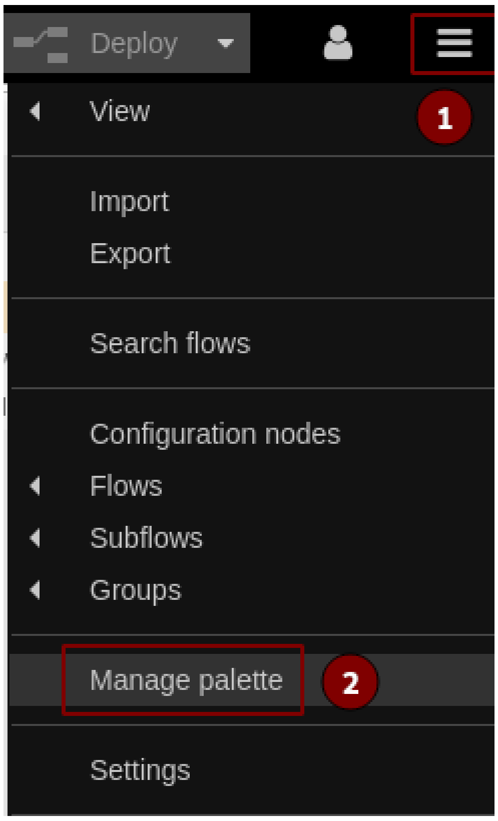


Figure 7: Install library

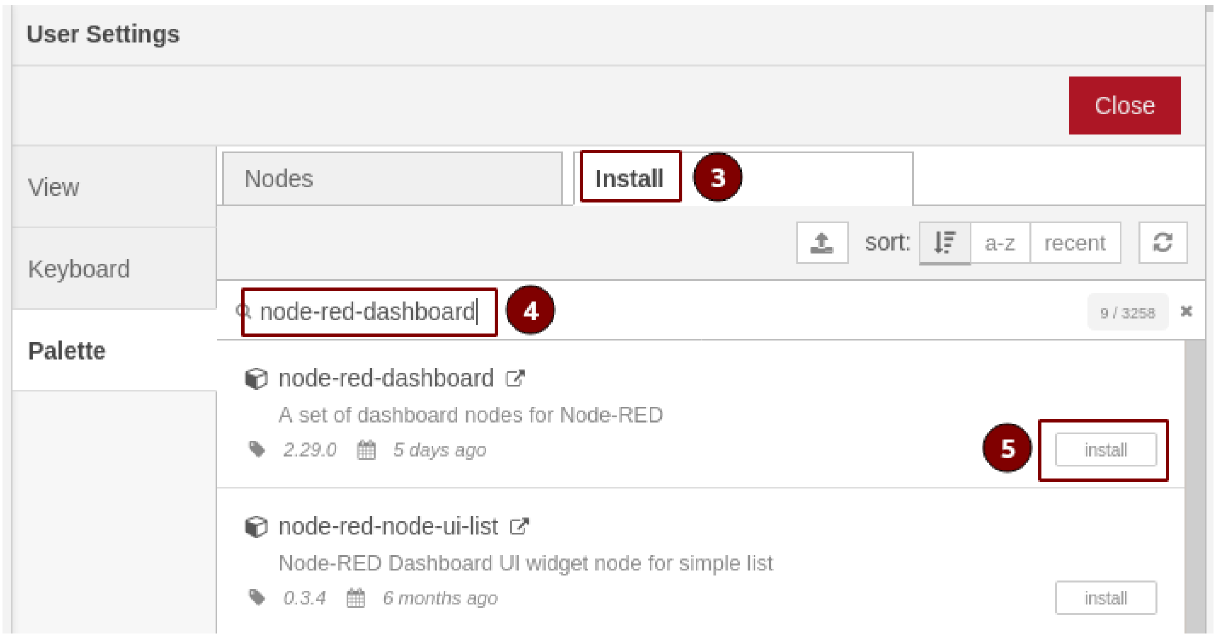


Figure 8: Install library

Import Demo AS flows We need to import the flows provided together with the LoRa Basics Modem.

First, import the first flow:

1. Click the menu in the top right corner of the screen.
2. Click `Import`.
3. Choose file `tools/application_server/basic-modem-as-flow1.json`.
4. Select `current flow`, and click `Import`.

Then, import `tools/application_server/basic-modem-as-flow2.json` follow above steps, but on the 4th step, replace `current flow` with **`new`** `flow` before click `Import`.

Below pictures shows how to do it:

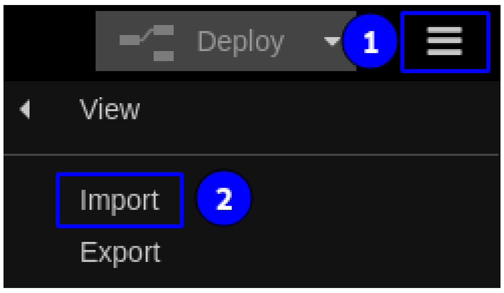
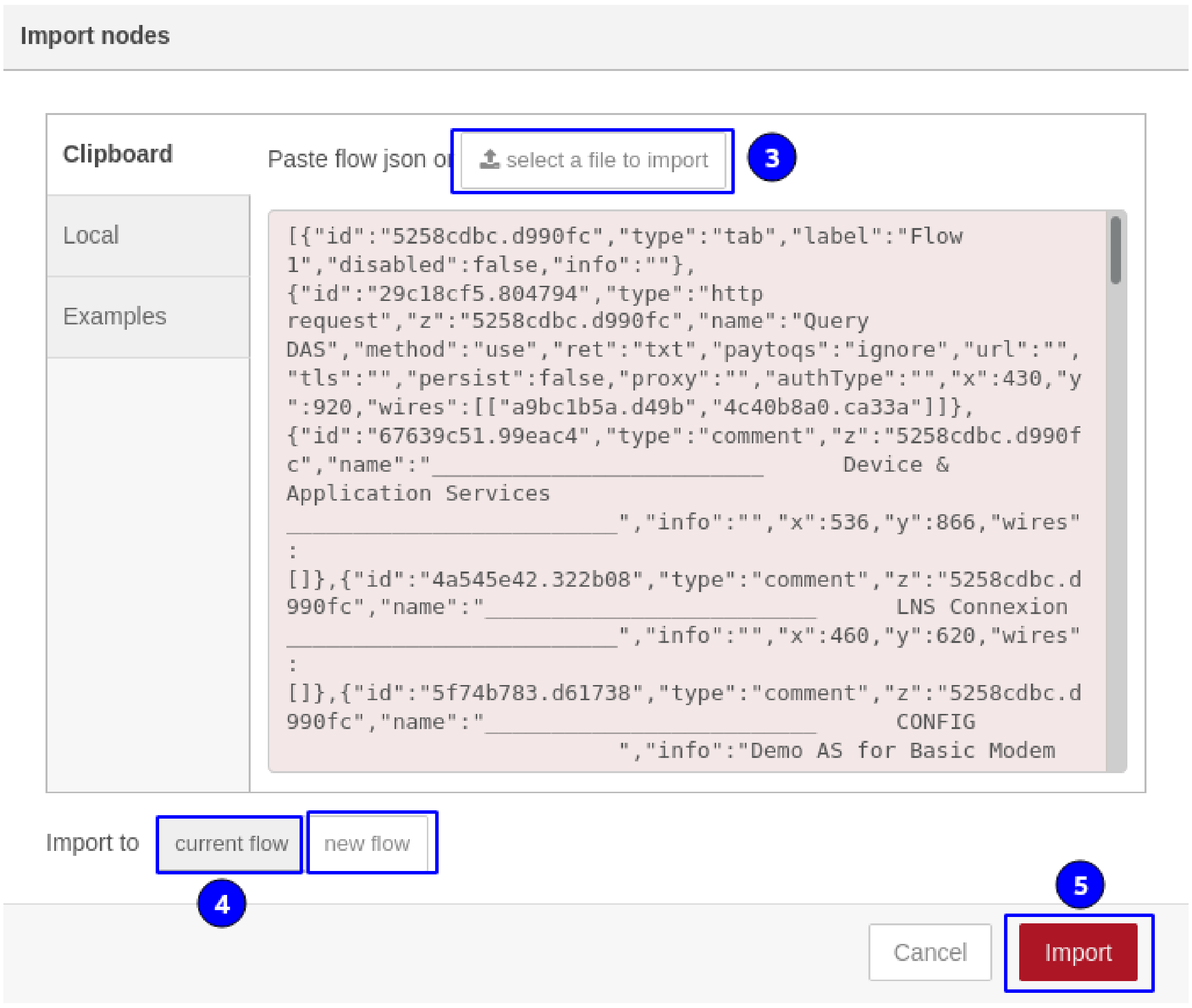


Figure 9: Import flows



```
[{"id":"5258cdbc.d990fc","type":"tab","label":"Flow
1","disabled":false,"info":""},
{"id":"29c18cf5.804794","type":"http
request","z":"5258cdbc.d990fc","name":"Query
DAS","method":"use","ret":"txt","paytoqs":"ignore","url":"",
"tls":"","persist":false,"proxy":"","authType":"","x":430,"y
":920,"wires":[["a9bc1b5a.d49b","4c40b8a0.ca33a"]]},
{"id":"67639c51.99eac4","type":"comment","z":"5258cdbc.d990f
c","name":"_________________________        Device &
Application Services
_________________________","info":"","x":536,"y":866,"wires"
:
[]},{"id":"4a545e42.322b08","type":"comment","z":"5258cdbc.d
990fc","name":"_________________________      LNS Connexion
_________________________","info":"","x":460,"y":620,"wires"
:
[]},{"id":"5f74b783.d61738","type":"comment","z":"5258cdbc.d
990fc","name":"_________________________      CONFIG
                         ","info":"Demo AS for Basic Modem
```

Figure 10: Import flows

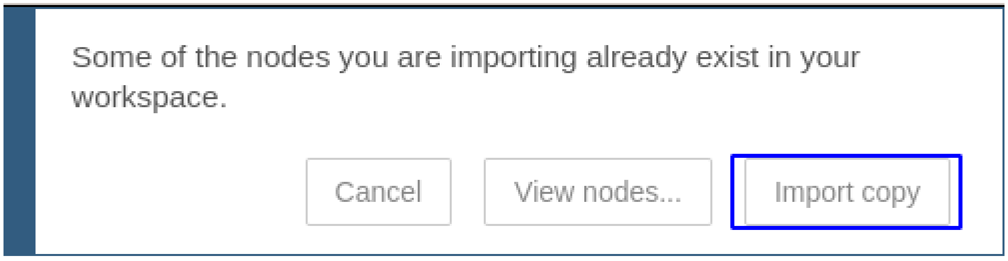

Some of the nodes you are importing already exist in your workspace.

Figure 11: Import flows

After that, we better remove the blank "Flow 1". Just double click it, then `Delete` it, as below picture shows:

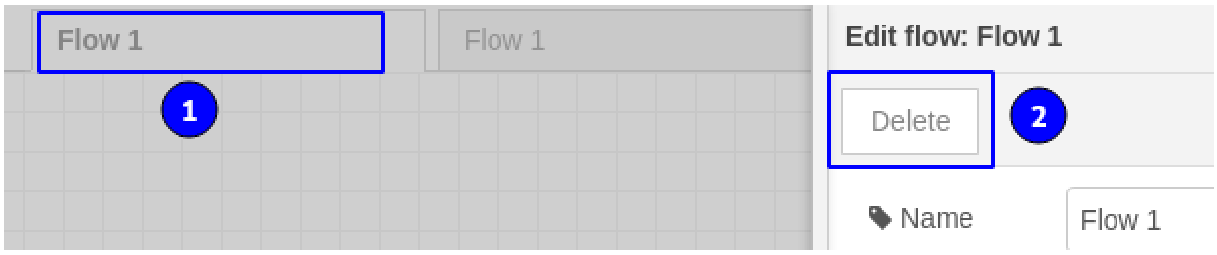


Figure 12: Remove blank Flow

And make sure Flow 1 is linked to Flow 2, as below picture shows:

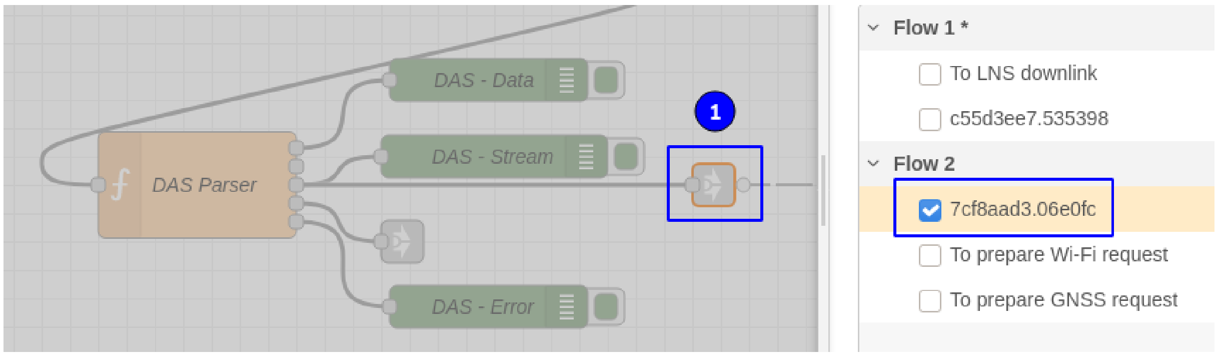
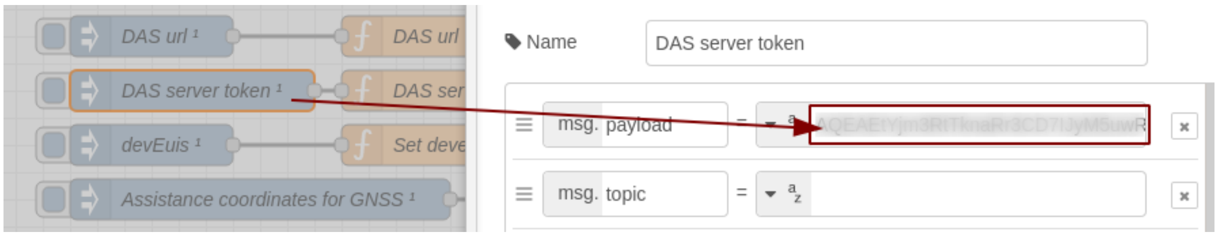


Figure 13: Link Flow

Set up Demo AS A few places need to be updated first before this AS works:

- Input user's DAS Token, and add Node DevEUI to the filter list.

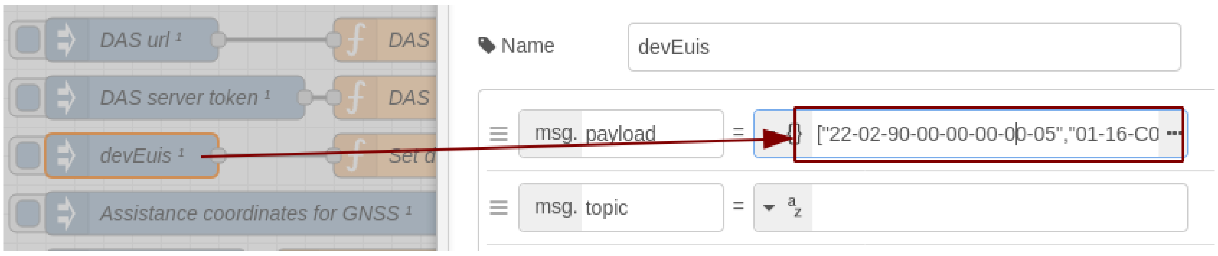


Name: DAS server token

msg. payload = [redacted]

msg. topic =

Figure 14: update token



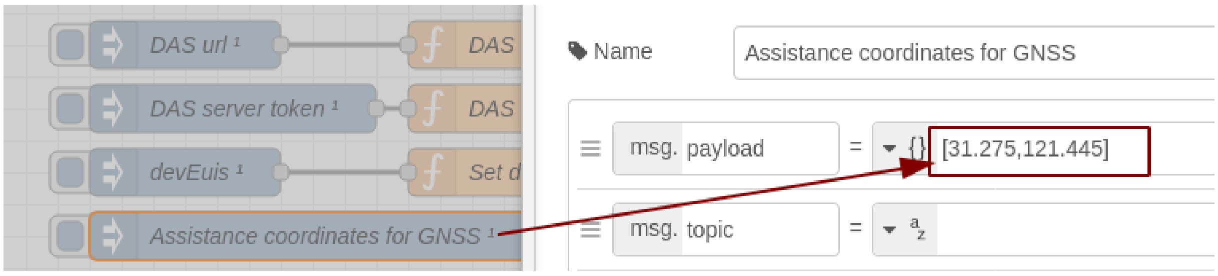
Name: devEuis

msg. payload = ["22-02-90-00-00-00-00-05","01-16-C0

msg. topic =

Figure 15: update deveui

- Correct the latitude and longitude values for assisted GNSS scan.



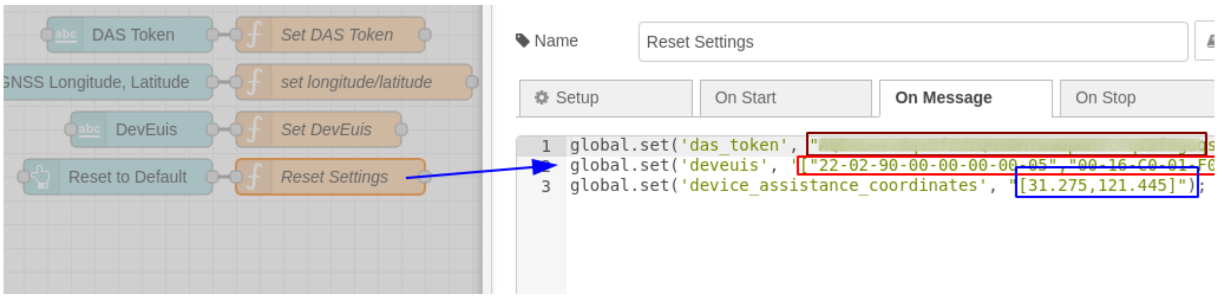
Name: Assistance coordinates for GNSS

msg. payload = [31.275,121.445]

msg. topic =

Figure 16: update geo

- Update the Token, DevEUI and latitude/longitude values for the function `Reset Settings`.



Name: Reset Settings

```
global.set('das_token', "[redacted]
global.set('deveuis', '["22-02-90-00-00-00-00-05","00-16-C0-01-F0
global.set('device_assistance_coordinates', "[31.275,121.445]");
```

Figure 17: update geo

Finally, to make the Log functions work all right, we need to change the `settings.js` a little:

- Create a folder under user's `HOME` folder named `logs`, then uncomment and change `httpStatic` in `settings.js` as:

```
    httpStatic: '/home/user/logs/',  (replace 'user' accordingly)
```

And add a few lines to section `functionGlobalContext` as below:

```
    functionGlobalContext: {
        fs:require('fs'),
        cryptojs:require('crypto-js'),
        aescmac:require('node-aes-cmac'),
        lorapacket:require('lora-packet'),
    },
```

Please be noted that this AS is based on the open source lorawan-server, with a MQTT server set up for it.

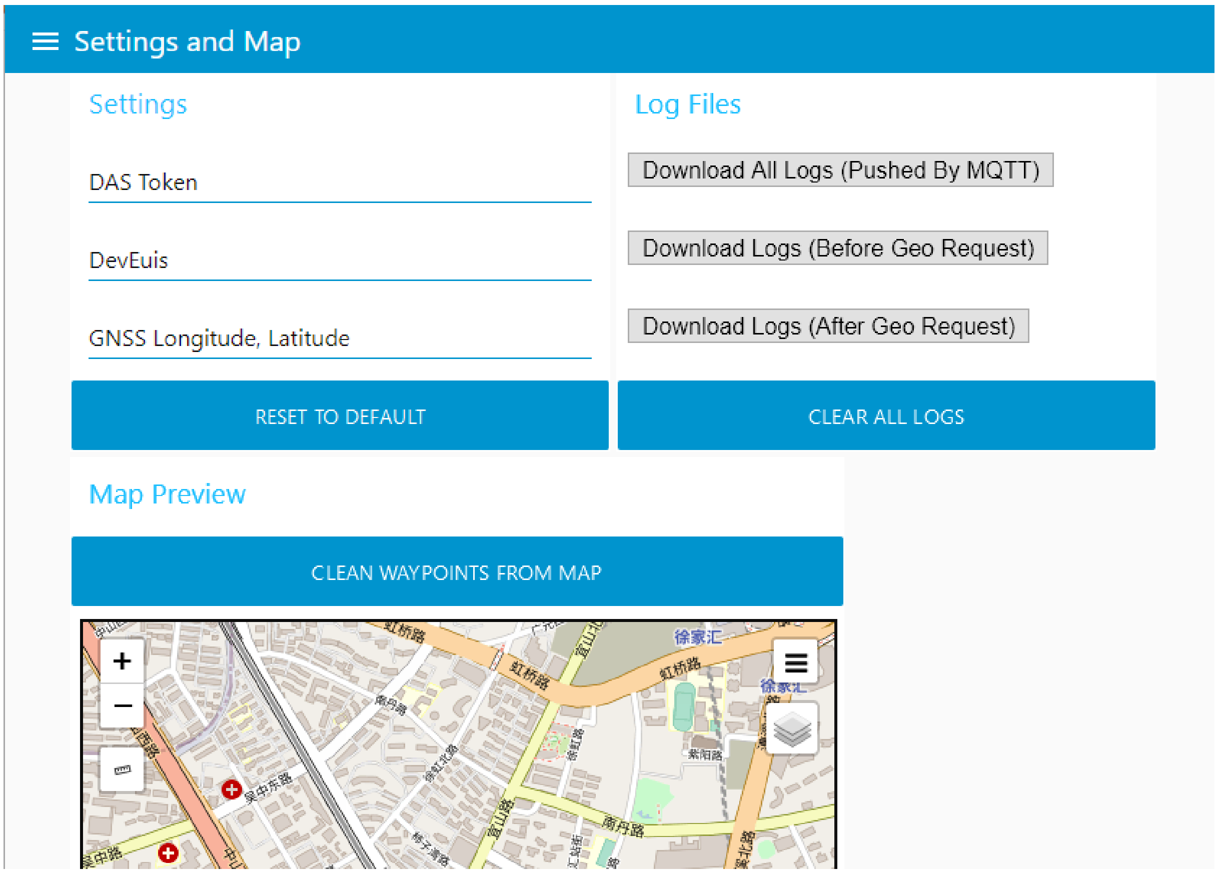
If user chooses other NS, he/she needs to change the `from NS` and `DAS Parser` parts accordingly.

The guide mentioned earlier at the beginning of this section contains useful information for support other NS.

Usage Guide

Once your AS is running, you can access it from: `http://<as_ip>:<as_port>`. (The default port is `1880`). On that page, you can make whatever modification you like.

As for the web UI interface, go to this address: `http://<as_ip>:<as_port>/ui`. The page looks like this:



Settings and Map

Settings

DAS Token

DevEuis

GNSS Longitude, Latitude

RESET TO DEFAULT

Log Files

Download All Logs (Pushed By MQTT)

Download Logs (Before Geo Request)

Download Logs (After Geo Request)

CLEAR ALL LOGS

Map Preview

CLEAN WAYPOINTS FROM MAP

Figure 18: Application Server UI

There are four items for configuration:

1. `DAS Token`: This item can be empty if the right token has been embedded in the source code page.
2. `DevEuis`: You need to register your Node's DevEui to it as a filter, with format similar as ["00-00-00-00-aa-bb-cc-dd","11-22-33-44-55-66-77-88"], otherwise all packets will be dropped.
3. `GNSS Longitude, Latitude`: You need to provide a rough value of your Node's longitude and latitude values to it. Those values are for GNSS Assisted Scan Mode.
4. `RESET TO DEFAULT`: Any value you entered to above 3 text field would be applied automatically, so if you entered some wrong values, you can use this button to restore to the default DAS Token and default longitude/latitude (which should be changed first to user's own values in the source code page), then make other changes if necessary.

If everything goes all right, then the Tracker try to join to NS once it boots. After the join is successful, it starts to do the Wi-Fi Passive Scan.

As for GNSS Scan, it needs to wait for the AS to send an accurate GPS time first, and after it receives the time, it will begin to do the GNSS Scan, and stream the GNSS data to AS if it can find any satellites.

The maximum number of satellites it sends to AS is 15, and maximum Wi-Fi SSID number is 10.

Once the AS gets such data from NS by MQTT, it requests the geolocation from DAS Solver, and shows the position on the map, as below:

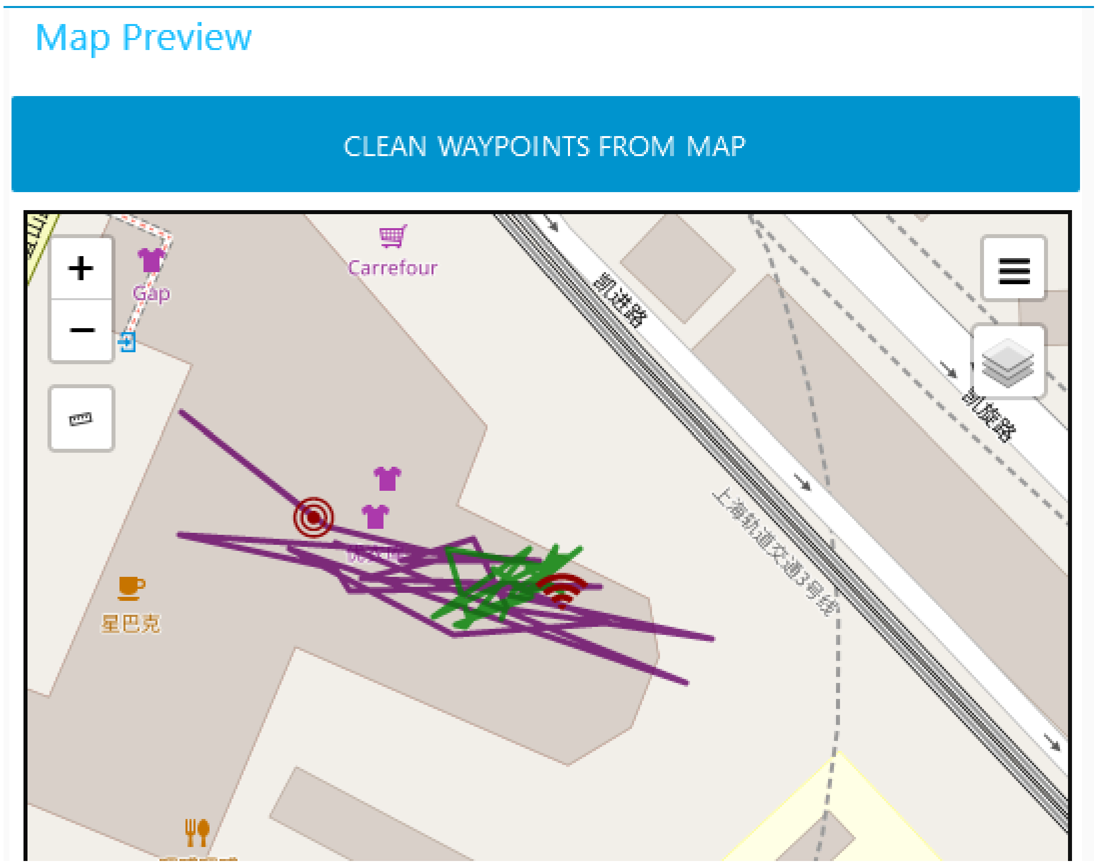


Figure 19: Waypoints on Map

Both Wi-Fi and GNSS waypoints on the map are connected by their own line to show the trace.

The waypoints will be kept on the map for 24 hours, but you can clean them using the button `CLEAN WAYPOINTS FROM MAP` above the map.

You can also view the full-page map from this address: `http://<as_ip>:<as_port>/worldmap`.

There are also buttons used for downloading the logs. There are three kinds of log as below picture shows:

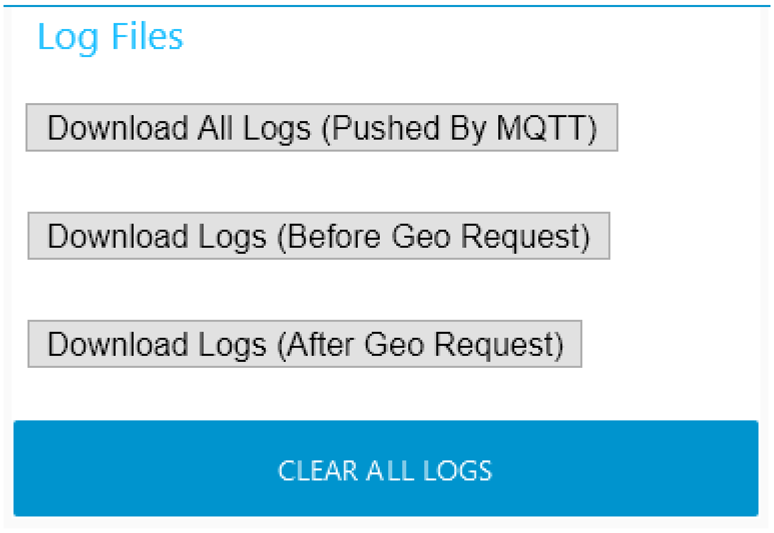

Log Files
Download All Logs (Pushed By MQTT)
Download Logs (Before Geo Request)
Download Logs (After Geo Request)
CLEAR ALL LOGS

Figure 20: Download Log Files

1. Logs pushed by MQTT from NS. This file saves all packets pushed from NS using MQTT.
2. Logs before Geo Request. This file saves all packets preparing for requesting the geolocation position, including both Wi-Fi and GNSS packets.
3. Logs after Geo Request. This file saves the results of geolocation request, so you will find the longitude/latitude values, or `logmsg` tells what goes wrong if the Solver does not provide the geolocation results.

Note: These logs contain a lot of information, but it requires some work to extract the useful information from it and get conclusions such as PER, or RSSI charts.

You can clean the logs from AS using the button `CLEAR ALL LOGS`. It will not ask for confirmation, so make sure this button is not clicked by accident.

Some Notes

- Each free TOKEN only supports 200 valid Wi-Fi requests to Solver per day. Once exceeded, the "After Req" log file will contain such information at the beginning of each line:

  {"payload":{"result":{"00-16-C0-01-F0-00-17-B9":{"error":"owner-::4ce:00-00-00-00-12-34-56-01: wifi limit exceeded: requested 1 + current 200 > limit 200"}}}

Change History

Latest version: V0.6: @ 2021.04.29

Change History

- V0.1: The initial draft.
- V0.2: Changes accordingly following the code changes based on the code review
- V0.3: Change to html and pdf format based on Markdown
- V0.4: Heavily changed based on peer review
- V0.5: Updates for Application Server installation
- V0.6: Add content