



## Building a WiFi localisation node

### Identifying the components

Our bill of materials (Complete Workshop set available at [TinyTronics](#))

- RFM Module RFM95W (868 Mhz)
- Wemos Lora PCB (<https://github.com/hallard/WeMos-Lora> )
- Wemos D1 mini v3 – ESP8266
- Micro USB cable

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

## Starting up the Arduino Environment

**You may skip this step if you have:**

- **Arduino installed with ESP8266 support**
- **LMIC 'Charles' version installed**

Setup the Arduino Environment. To use the ESP8266 we need the ESP toolchain. Installation instructions can be found here: <https://github.com/esp8266/Arduino/>

## Installing with Boards Manager

Starting with 1.6.4, Arduino allows installation of third-party platform packages using Boards Manager. We have packages available for Windows, Mac OS, and Linux (32 and 64 bit).

- Install the current upstream Arduino IDE at the 1.8 level or later. The current version is at the [Arduino website](#).
- Start Arduino and open Preferences window.
- Enter `http://arduino.esp8266.com/stable/package_esp8266com_index.json` into *Additional Board Manager URLs* field. You can add multiple URLs, separating them with commas.
- Open Boards Manager from Tools > Board menu and install *esp8266* platform (and don't forget to select your ESP8266 board from Tools > Board menu after installation).

Close your Arduino IDE software. Remove (if already installed) the current LMIC library. We will use the 'Charles' version of the library. This one is compatible with your other projects, but has some special functionality needed for the WeMos LoRa board (supporting one input pin for the three DIO pins).



Adafruit_NeoPixel	11-11-2017 / 19:58	Bestandsm
Adafruit_SSD1306	13-12-2016 20:49	Bestandsm
Adafruit_Unified_Sensor	27-5-2017 11:56	Bestandsm
ArduinoJson	11-12-2017 21:38	Bestandsm
arduino-lmic-master	13-5-2017 11:43	Bestandsm
AzureIoT_Hub	11-11-2017 19:58	Bestandsm
AzureIoTProtocol_HTTP	11-11-2017 20:05	Bestandsm
AzureIoTProtocol_MQTT	11-11-2017 19:58	Bestandsm

Download the new library and install the .zip file with library manager:

1. Goto <https://github.com/ch2i/arduino-lmic>
2. Choose 'Clone or download'
3. Download ZIP
4. Mark the location
5. Go to Arduino IDE
6. Select 'Sketch->Include Library->Add .ZIP Library'
7. Select the downloaded Arduino-lmic-master.zip file from your download location

Change the 'lmic\_project\_config.h' file in the project\_config folder (remove the remark from CFG\_eu868 and add them to CFG\_us915):

```
//#define CFG_us915 1
#define CFG_eu868 1
#define CFG_sx1276_radio 1
```

Download the Sketch from [https://github.com/galagaking/Wemos\\_Node/WiFiLocalisation.ino](https://github.com/galagaking/Wemos_Node/WiFiLocalisation.ino)

## The Things Network Dashboard

Your applications and devices can be managed by The Things Network Dashboard.

### Create an Account

To use the dashboard, you need a The Things Network account. You can create an account here:

<https://account.thethingsnetwork.org/users/login> .

After registering and validating your email address, you will be able to log in to The Things Network Dashboard.

### Create an Application

<https://console.thethingsnetwork.org/applications>

Choose 'add application'

Give your Application a unique ID. You can use ONLY lowercase! You can add an unique number to get uniqueness over the ttn network (this is a global ID). If you already use an application you better add another one, because of the specific Payload Function needed here.

Your description can be any description you like.



**ADD APPLICATION**

**Application ID**  
The unique identifier of your application on the network

fb\_nodes

**Description**  
A human readable description of your new app

My experimental nodes

☒ **Register application on the handler ttn-handler-eu**  
If selected, registers your app to the handler ttn-handler-eu, to make it usable right away

Cancel Add application

## ABP

Activation by Personalization (ABP) is a method where the security keys are stored in the device. Not as safe as the OTAA method, but for experiments it works OK. There is no join procedure, nodes will work right away.

**Device ID**  
This is the unique identifier for the device in this app. The device ID will be immutable.

abp\_node

**Device EUI**  
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

56 29 AA BB 56 29 CC DF

**App Key**  
The App Key will be used to secure the communication between you device and the network.

App Key will be generated

**App EUI**

70 B3 D5 7E D0 00 17 BE

Choose *Register the device*.

Now edit the settings of the device and choose ABP (OTAA will be selected by default)



Device EUI

56 29 AA BB 56 29 CC DF

App EUI

70 B3 D5 7E D0 00 17 BE

Activation method

OTAA

ABP

Device Address

The device address will be assigned by the network server and is not customizable

Select 'save'. Now some values are system generated and we have to copy them to our code. Get the code from [https://github.com/galagaking/Wemos\\_Node](https://github.com/galagaking/Wemos_Node) We use the Wemos\_Node.ino example here.

Device Address

<> 26 01 1A 32 hex

Network Session Key

<> { 0x21, 0xEE, 0x1E, 0x77, 0x58, 0xBD, 0xFE, 0x47, 0x45, 0x34, 0x msb

App Session Key

<> { 0xFE, 0xC6, 0xE8, 0x6E, 0x4D, 0x31, 0x35, 0x4D, 0xA5, 0x65, msb

- Copy the Device Address as a HEX value to DEVADDR in the example, so 26 01 1A 32 will be 0x26011A32.
- Copy the Network Session Key as MSB to NWSKEY.
- Copy the App Session Key as MSB to APPSKEY.

```
// LoRaWAN NwsKey, network session key, AppSKey, application session key, end-device address
static const PROGMEM u1_t NWSKEY[16] = { 0x21, 0xEE, 0x1E, 0x77, 0x58, 0xBD, 0xFE, 0x47, 0x45, 0x34, 0x
static const PROGMEM u1_t APPSKEY[16] = { 0xFE, 0xC6, 0xE8, 0x6E, 0x4D, 0x31, 0x35, 0x4D, 0xA5, 0x65, 0xA
// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26011A32 ; // <-- Change this address for every node!
```

Select the Wemos board in the Arduino IDE (no debug port, lwIP v2 Prebuilt, CPU 80 Mhz, Upload speed 921200)

Board: "WeMos D1 R2 & mini"

Flash Size: "4M (1M SPIFFS)"

Debug port: "Disabled"

Debug Level: "None"

lwIP Variant: "v2 Prebuilt (MSS=536)"

CPU Frequency: "80 MHz"

Upload Speed: "921600"

Port

v2 Prebuilt (MSS=536)

v2 Prebuilt (MSS=1460, unstable)

v1.4 Prebuilt

v1.4 Open Source



Compile and upload the code. Check in the dashboard the working.

You might want to uncheck the frame counter check

Frame counter width

16 bit 32 bit

☒ Frame counter checks

If the frame counter is checked, you must respect the sequence number, and probably copied packages are refused on the network. Though restarting you node, and thereby resetting the frame counter, will disable your node. So to get this working, you have to disable this check by unchecking this box.

To get the right display format, we can create a decoder in our application. Select your application and open the 'Payload Functions':

PAYLOAD FUNCTIONS

decoder converter validator encoder

```
1 function Decoder(bytes, port) {
2   // Decode an uplink message from a buffer
3   // (array) of bytes to an object of fields.
4   var decoded = {};
5
6   // if (port === 1) decoded.led = bytes[0];
7
8   return decoded;
9 }
```

Enter the function below, overwriting the standard function

```
var hexChar = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"];

function byteToHex(b) {
  return hexChar[(b >> 4) & 0x0f] + hexChar[b & 0x0f];
}

function hexToInt(hex) {
  var num=hex;
  if (num>0x7F) {
    num=num-0x100;
  }
  return num;
}

function Decoder(bytes) {
```



```
var mac1="";
for (i = 0; i < 6; i++) {
    mac1 += byteToHex(bytes[i]);
    if (i<5) { mac1+=':';}
}
var rssi1=hexToInt(bytes[6]);
var mac2="";
for (i = 0; i < 6; i++) {
    mac2 += byteToHex(bytes[i+7]);
    if (i<5) { mac2+=':';}
}
var rssi2=hexToInt(bytes[13]);
return {
    macaddress: {
        mac_1: mac1,
        rssi_1:rssi1,
        mac_2: mac2,
        rssi_2:rssi2,
    },
};
}
```

[decoder](#)[converter](#)[validator](#)[encoder](#)

```
1 var hexChar = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"];
2
3 function byteToHex(b) {
4     return hexChar[(b >> 4) & 0x0f] + hexChar[b & 0x0f];
5 }
6 function hexToInt(hex) {
7     var num=hex;
8     if (num>0x7F) {
9         num=num-0x100;
10    }
11    return num;
12 }
```

First test the function with two dummy bytes CE D0, And then Save the function.

Return to your data and look what happens:



APPLICATION DATA									
Filters									
uplink   downlink   activation   ack   error									
time   counter   port									
▲ 22:12:14	11	1	dev id: <a href="#">wemos_01</a>	payload: 78 8A 20 4C 04 C8 D0 7A 8A 20 4C 04 C8 D0	macaddress.mac_1: "78:8A:20:4C:"				
▲ 22:11:06	10	1	dev id: <a href="#">wemos_01</a>	payload: 78 8A 20 4C 04 C8 D0 7A 8A 20 4C 04 C8 CF	macaddress.mac_1: "78:8A:20:4C:"				

The payload contains two BSSIDs and two RSSI values.

## Node Red

Now we have to decode these BSSID values to a position on a map. We will use Node Red and Google for this.

[Get a Google API key](#)

Go to <https://developers.google.com/maps/documentation/geolocation/get-api-key>

Enable Google Maps Geolocation API

My Project

CANCEL NEXT

Click 'Next'

Your API key is here, store it for future use:

You're all set!

You're ready to start developing with Google Maps Geolocation API

YOUR API KEY

AIzaSyCN7DfEK1qp1f

To improve your app's security, restrict this key's usage in the [API Console](#).

DONE

Create a new 'Flow' in Node Red with '+'



## Create an App in BlueMix

An App is a functionality installed on the BlueMix platform. In the main dashboard choose 'Create Resource'.

The screenshot shows the IBM Cloud Dashboard. At the top, there's a navigation bar with 'Catalog', 'Docs', 'Support', and 'Manage'. Below the navigation bar, the 'Dashboard' section is visible. It includes filters for 'RESOURCE GROUP' (All Resources), 'REGION' (US South), 'CLOUD FOUNDRY ORG' (TTN\_Environment), and 'CLOUD FOUNDRY SPACE' (TTN\_US). A 'Create resource' button is highlighted with a red circle. Below the filters, there are two sections: 'Cloud Foundry Apps' and 'Cloud Foundry Services'. The 'Cloud Foundry Apps' section shows a table with columns: Name, Route, Memory (MB), and State. It lists two apps: 'java-cloudant-form-geojson-example-20171113213230447' (Stopped) and 'TTNWorkshop' (Running). The 'Cloud Foundry Services' section shows a table with columns: Name, Service Offering, and Plan. It lists one service: 'availability-monitoring-auto' (Availability Monitoring, Lite).

In the next screen we choose 'Node-Red Starter'

The screenshot shows the IBM Cloud Catalog. On the left, there's a sidebar with 'All Categories' and a list of categories: Infrastructure (Compute, Storage, Network, Security, Containers, VMware), Platform (Boilerplates, APIs, Application Services, Blockchain, Cloud Foundry Apps, Data & Analytics). The main area displays various starters. The 'Node-RED Starter' is highlighted with a red circle. It is described as 'This application demonstrates how to run the Node-RED open-source project'. It has 'Lite' and 'Community' versions available. Other starters visible include 'MobileFirst Services Starter', 'Node.js Cloudant DB Web Starter', 'Personality Insights Java Web Starter', 'Personality Insights Node.js Web Starter', 'StrongLoop Arc', 'Mendix Rapid Apps', 'Python Flask', and 'Vaadin Rich Web Starter'.

Select 'Node-RED Starter':

Now you need a hostname for your Node-Red server instance, so this should be globally unique! The App name is just unique for your own environment. Be creative!





← View all

## Create a Cloud Foundry App

---

### Node-RED Starter

This application demonstrates how to run the Node-RED open-source project within IBM Bluemix.

**Life** **Community**

[View Docs](#)

VERSION 0.7.0  
TYPE Boilerplate  
REGION United Kingdom, Germany, US South, Sydney

**App name:**  
My-Unique-Name

**Host name:**  
My-Unique-Name

**Domain:**  
eu-gb.mybluemix.net

**Choose a region/location to deploy in:**  
United Kingdom

**Choose an organization:**  
TTN\_Environment

**Choose a space:**  
dev


**Selected Plan:**  
SDK for Node.js™ Cloudant NoSQL DB

Choose 'Create', this will create a Node-RED instance.

Your Node-RED server will be created and started. It can last up to 5 minutes!

After the creation you can select the 'app url':

Cloud Foundry apps /



# My-Unique-Name

Starting[Visit App URL](#)

**Org:** TTN\_Environment **Location:** United Kingdom **Space:** dev

A screen appears, asking you to secure your Node-Red editor

## Welcome to your new Node-RED instance on IBM Bluemix

We know you're eager to start wiring up your flows, but first there are a couple of tasks you should do:

- Secure your Node-RED editor
- Browse available IBM Bluemix nodes

Previous

Next

Enter (and remember) a username and password



## Secure your Node-RED editor

☒ Secure your editor so only authorised users can access it

Username

Password

☐ Allow anyone to view the editor, but not make any changes

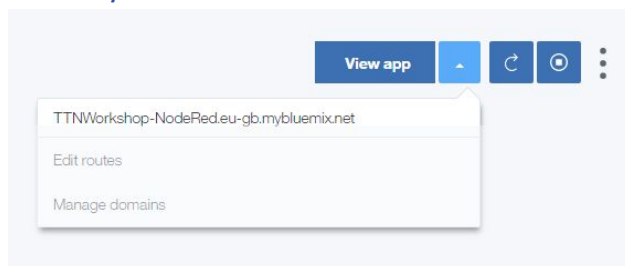
☐ *Not recommended:* Allow anyone to access the editor and make changes

If you do not enter any password, your Node-Red instance will be worldwide open and available!

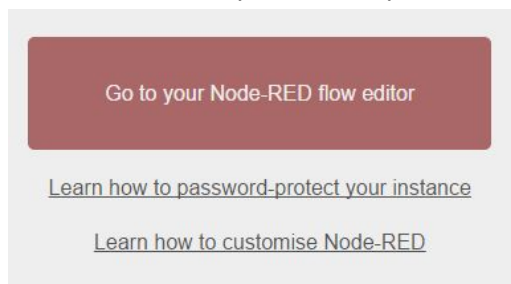
Skip the next info screen 'Browse available IBM Bluemix nodes'.

Now press 'Finish' to finish the installation.

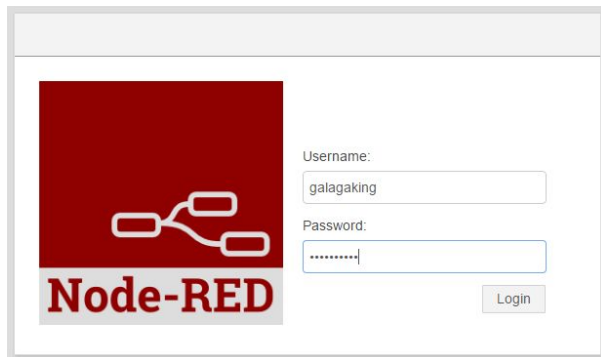
## Access your Node-Red server



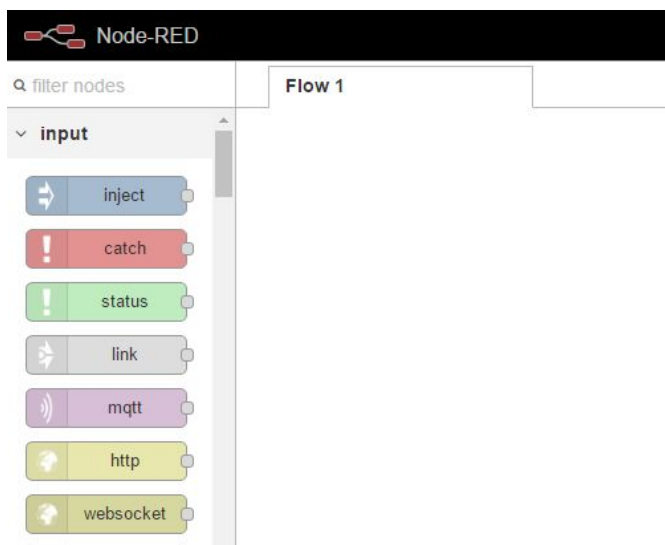
Select the name of your server, you can also use this URL to access your Node-Red server.



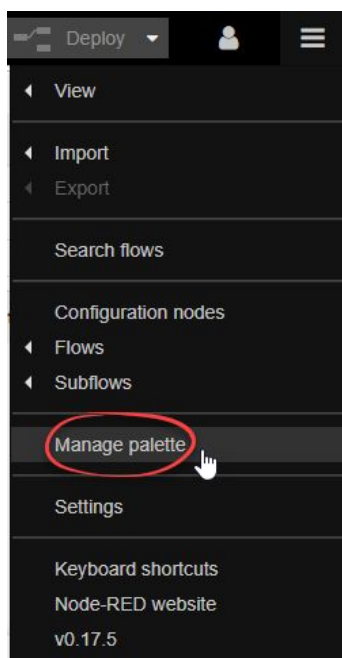
Now you have to enter your previous password:



Your Node-Red is nearly ready to roll!



Now select the top right menu:





Choose manage Palette

A screenshot of the "Manage palette" interface. It has a header with a "Done" button. Below is a tabbed interface with "Nodes" and "Install" tabs. The "Install" tab is active, showing a search bar with "ttn" entered, a "sort" dropdown set to "a-z", and a "recent" button. Below the search bar, a list of nodes is shown, with "node-red-contrib-ttn" selected. The node details show "The Things Network Node-RED Application Nodes", version "2.0.0", and "3 months ago". An "install" button is visible next to the node details.

- Search for TTN in the Install tab and click install to install the TTN nodes.
- Click again install on the warning screen.
- Search for Worldmap, and install.
- 'Done' to exit the palette management.

TTN nodes can now be added in your Node-Red flow.

## Node Red introduction

We start with a clean 'flow'. Remove any flows with the Menu->Flows->Delete function.

Add a TTN message node by drag and drop it on your flow screen.

By double clicking on the TTN message node you can change the values:

A screenshot of the "Edit ttn message node" interface. It has a header with "Delete", "Cancel", and "Done" buttons. Below are four input fields: "Name" with the value "sensor", "App" with a dropdown menu showing "fb\_nodes" and a pencil icon, "Device ID" with the value "sensor\_04", and "Field" with the value "celcius".

The name can be any name,

App refers to your TTN console. With the Pencil you can add your application here:

ttn message > **Edit ttn app node**

fb\_nodes

eu

.....

- App ID is the written name 'Application ID' from the TTN console
- Region is eu (you have to fill this in!!)
- Access Key is a copy of the access key of your application (access key of application, default key).
- Check Update and select your App in the previous screen.
- Device ID is the Device ID in TTN console

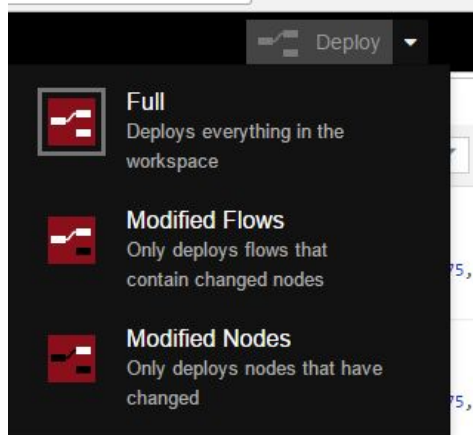
You can select fields you want to have as an input for your flow, the names depend on your Payload Function. If you do not have a payload function loaded, you can use an empty 'field', all the output will be shown.

Now add a 'Debug' output node to your flow and connect the both with a wire:



Select msg.payload by double clicking on the debug node and activate it by clicking on the button on the right.

Now activate your flow with Deploy:



You will see the results of your node appear in the right debug column.



## Google Geolocation

Now will will create a request to Google based on these MAC addresses and our Google API key.

Create a 'Function' object with the function 'AskGoogle.js' on Github (copy / paste from this text will NOT work!).

```
var google_apikey = "FILL_IN_YOUR_GOOGLE_API_KEY";  
  
var dev_url =  
"https://www.googleapis.com/geolocation/v1/geolocate?key="+google_apikey;  
  
var data =  
{  
  "considerIp": "false",  
  "wifiAccessPoints": [  
    {  
      "macAddress": msg.payload.macaddress.mac_1,  
      "signalStrength": msg.payload.macaddress.rssi_1,  
      // "signalToNoiseRatio": 0  
    },  
    {  
      "macAddress": msg.payload.macaddress.mac_2,  
      "signalStrength": msg.payload.macaddress.rssi_2,
```



```
// "signalToNoiseRatio": 0
}
]
};
var msg = {
  "method" : "POST",
  "url" : dev_url,
  "headers" : {
    "Content-Type": "application/json",
    // "X-M2X-KEY": google_api_key
  },
  "payload" : JSON.stringify(data)
};

return msg;
```

In the first line, fill in your Google API Key.

Connect this function with the node.

The next node is a HTTP request node. Because we created our request in a function, this one has just the Method: -set by msg.method-. All the parameters will be filled in by the msg object.

Edit http request node

Delete Cancel Done

node properties

Method: - set by msg.method -

URL: http://

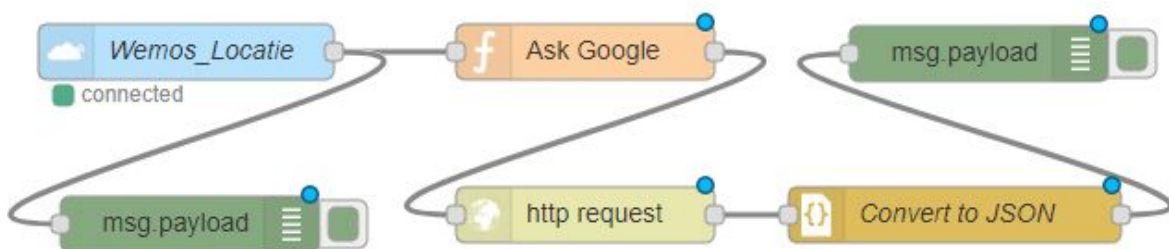
☐ Enable secure (SSL/TLS) connection

☐ Use basic authentication

Return: a UTF-8 string

Name: Name

To convert the Google Output back to JSON we will just add a JSON function. Add a Debug node to show the results.



You will see the debug flow running. Google will respond with your location (by clicking on the

arrows you open the objects), showing Lat and Lng and accuracy.

```

msg.payload : Object
└─ { macaddress: object }

17-1-2018 22:37:11 node: a7ef8532.5812d8
msg.payload : Object
▼ object
  ▼ location: object
    lat: 51.487027399999995
    lng: 5.4823499
    accuracy: 20

```

## WorldMap

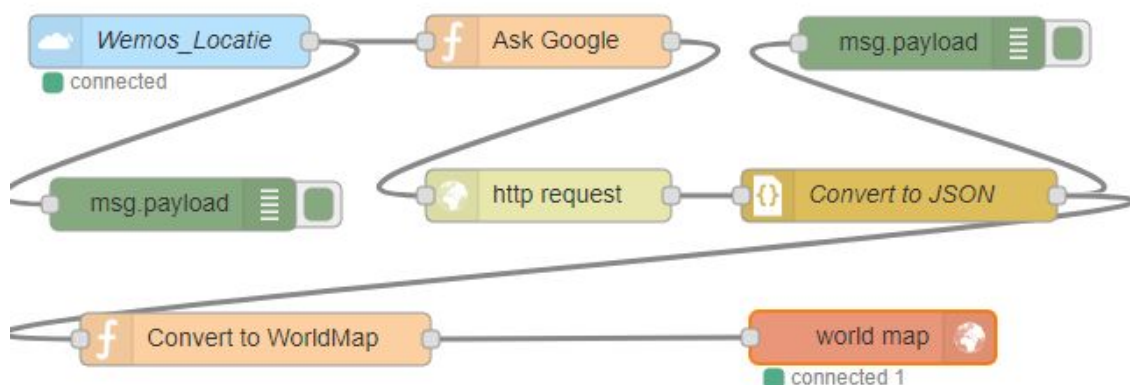
To display the info we install 'WorldMap' from the palette. WorldMap needs a different input format so we have to convert this by adding a function 'PutWorldMap.js' on github:

```

var msg1 = {};
msg1.payload=msg.payload;
msg1.payload.lon= msg.payload.location.lng;
msg1.payload.lat=msg.payload.location.lat;
delete msg.payload.location;
msg1.payload.name="Wemos";
msg1.payload.icon = "map-pin";
msg1.payload.iconColor = "orange";
msg1.payload.radius=msg.payload.accuracy;
msg1.payload.date=Date().toString();
return msg1;

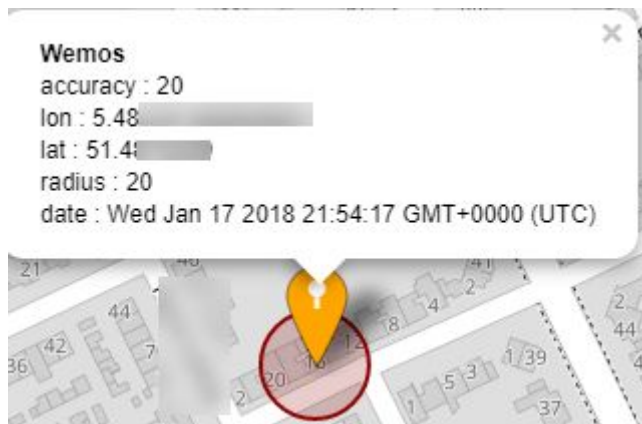
```

We add a color, a circle for the accuracy and a date. Our flow will look like this:

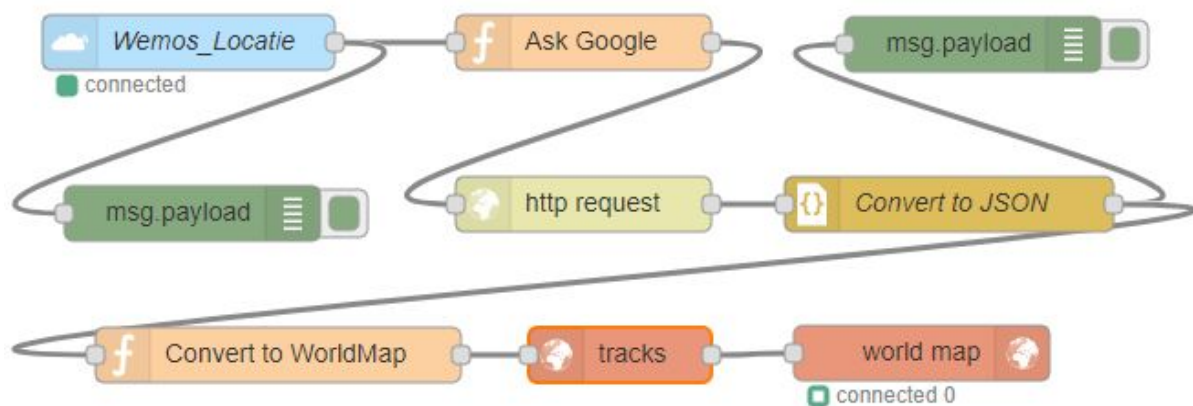


To show the 'WorldMap' press CTRL-SHIFT-M, or add '/worldmap' to your NodeRed URL:





To keep track of your position you can add the 'Tracks' node:



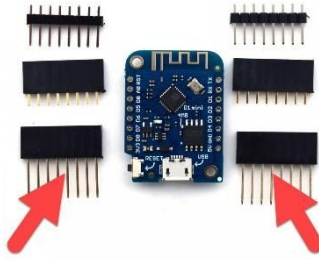
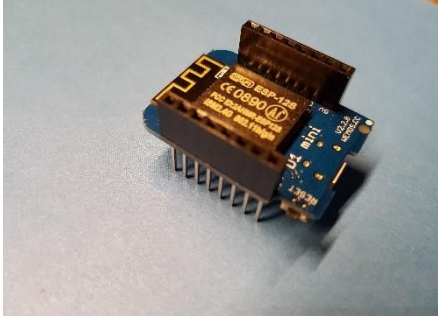
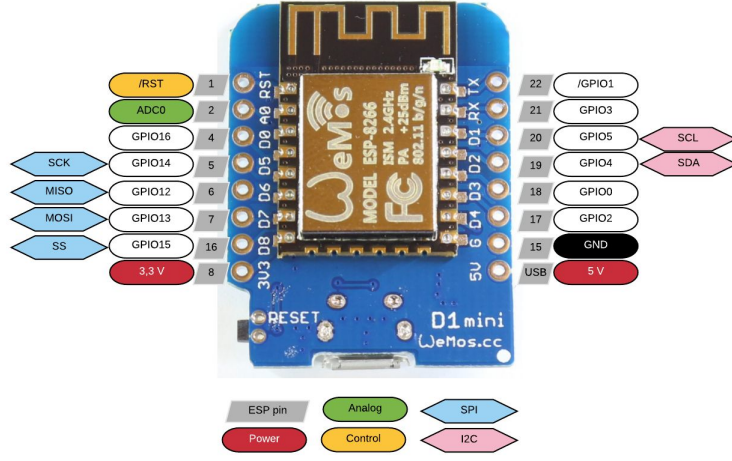
Thanks to Charles-Henri Hallard / Ch2i for his cooperation and sharing his PCB design for workshop purposes.

Frank Beks

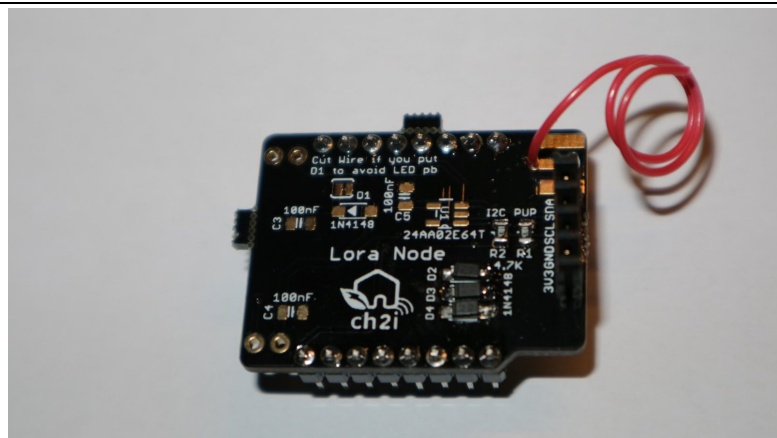
February 2017

## The Wemos Stack

Wemos (<http://www.wemos.cc>) offers a broad range of 'stackable' modules (displays, LEDs, buttons, power and processor boards) to be 'stacked' to your favourite application. We will use the Wemos D1 ESP8266 module and the 'WeMos-Lora' board. This offers a module which can connect to both WiFi networks as to LoRaWAN. As long as the right pins are connected, you can 'stack' it in different ways. We will use the experimental setup with the 'through-the-hole' pins.

<p>Use the 'long' headers</p>	
<p>The processor will be on top, the switch on the bottom right with 3v3 power on the left and 5v power on the right pin.</p>	
<p>Pinout looking on the top of the module, 3v3 on the lower left, 5v on the lower right.</p>	

Solder the two headers on the same side as the RFM95 radio, pins facing down. Mount a 868 Mhz Antenna (82mm) on the antenna pin. If you do not use the pre-soldered version, you have to mount RFM95, D2, D3, D4 (1n4148), R1, R2 (4k7), C1 (1uF) and C2 (10uF). You can use the i2c bus connector or add the ws2812 LEDs as well, but they are not used here.



Cut a piece of wire, length 82.2 mm. This is a  $\frac{1}{4}$  wave length antenna.

Wavelength =  $300/868 = 0.3456\text{m}$   
Wavelength / 4 =  $0.0864\text{ m}$   
correction  $0.95 \cdot 0.0864 = 0.0822\text{m}$  (to determine the precise length you can do some experiments, or use proper test equipment like a SWR meter)

Now you can 'stack' the two modules as shown. Mark the 'notch' on the lower right corner!

