

RN2384 hook-up guide (on Cibicom/Loriot server)

- using ESP8266

Setting up the ESP8266

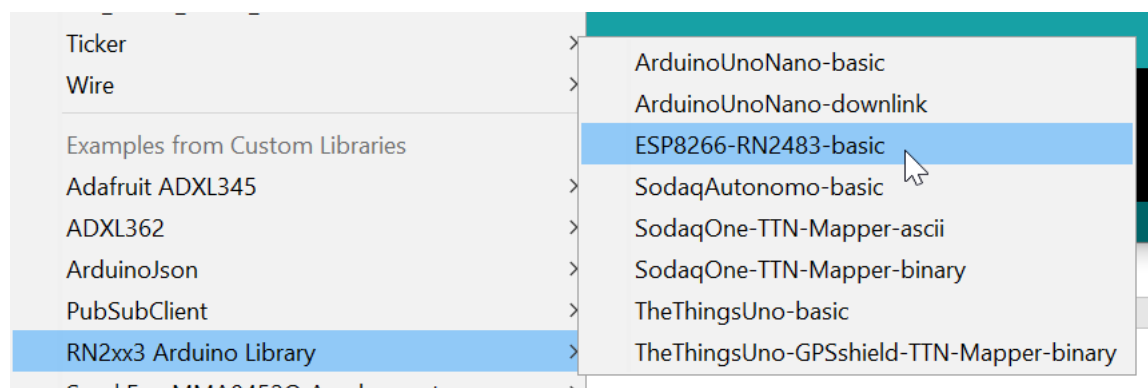
1. Install the RN2483 Library: <https://github.com/jpmeijers/RN2483-Arduino-Library> in your Arduino IDE. If unsure how to install a library use instructions here: <https://www.arduino.cc/en/Guide/Libraries>
2. Make the following connections between your ESP8266 and RN2483 module. Its recommended to place both modules in a breadboard.

ESP8266 **GPIO5** <-----> **RX** on RN2483
ESP8266 **GPIO4** <-----> **TX** on RN2483
ESP8266 **3.3V** <-----> **3.3V** on RN2483
ESP8266 **GND** <-----> **GND** on RN2483
ESP8266 **GPIO15** <-----> **RST** on RN2483

Note: Look up the NodeMCU pinout if unsure which pins are GPIO4, GPIO5 and GPIO15.

Question: From looking at the NodeMCU pinout and finding the connections GPIO4 and GPIO5 you are able to figure out what kind of communication protocol is used to communicate between the ESP8266 and RN2483. UART? I2C? SPI? Other?

3. Connect the antenna to the RN2483 board.
4. In the Arduino IDE, load example code *Examples -> RN2xx3 Arduino Library -> ESP8266-RN2483-basic*



Note that this example code is made for joining and transmitting to a TTN network (The Things Network).

5. Inspect the code and analyse what it does.
6. Before uploading the sketch make sure that you have connected the antenna. Upload and run the code and open the serial monitor. Remember to set the correct baud rate. You will find this in the code in Arduino IDE.
If you see an error message *"Communication with RN2xx3 unsuccessful. Power cycle the board."* double-check the connections. Also try to push the RST button on the ESP8266 module.

7. If the setup and connection is successful you should see something like this in the serial monitor:

```
Communication with RN2xx3 unsuccessful. Power cycle the board.
ct 31 2018 15:06:52
When using OTAA, register this DevEUI:
0004A30B00F2258B
RN2xx3 firmware version:
RN2483 1.0.5 Oct 31 2018 15:06:52
Trying to join TTN
Unable to join. Are your keys correct, and do you have TTN coverage?
```

From this output you will see the code and library have retrieved the hardware unique EUI number for the RN2483 module (DevEUI). This is a 16 hexadecimal characters identification number that we will need to add the device to the LoraWAN server (Cibicom server). The DevEUI was retrieved using this call

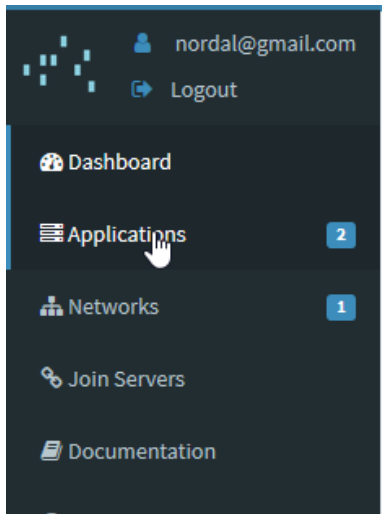
```
hweui = myLora.hweui();
```

Copy-paste the EUI number to a Notepad or similar. You will need it soon.

Note: You will also see a message that it was unable to join. This is because we have not inserted proper join credentials yet.

Joining the RN2483 to the Cibicom/Loriot network

- Log into the Cibicom / Loriot server with the credentials you have received (<https://iotnet.teracom.dk/>) and click “Applications”

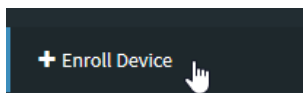


In Applications, use the existing application created called *IoT-course*. Notice the max devices allowed per application. This cannot be exceeded.

Network Applications

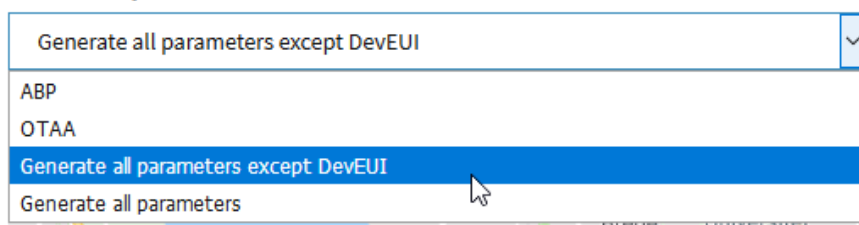
Applications				
	Name	App Id	Devices	Max. Devices
	Filter by name			
<input type="checkbox"/>	IoT-course	BE-7A-11-09	1	10

- Click on the application and look for “Enroll device” in the left navigation bar. Click the link.



- Now you can enrol your RN2483. Select the “Enrollment process”, in this case we will use Over the Air (OTAA) but we will let Loriot / Cibicom create the necessary Application EUI and Application Key. This is done by selecting “Generate all parameters except DevEUI”:

Enrollment process



Now insert the DevEUI that you got from the device before (HWEUI) and click Enroll:

Device Details

DevEUI

0004A30B0021BF-B3

☐ Create Another

Enroll

11. Now select your newly enrolled device in the device list:

	Device EUI	Name	RSSI (dBm)	SNR (dB)	devSNR (dB)	SF	BAT
	Filter per Device EUI						
<input type="checkbox"/>	00-04-A3-0B-00-21-BF-B3	00-04-A3-0B-00-21-BF-B3			N/A		N/A

From here we can get the Device address (DevAddr), which we don't need now, and the needed Application EUI (AppEUI) .

mnpe@fotonik.d... Logout

Back To Devices

DEVICE
00-04-A3-0B-00-21-5B-73

Location

LoRaWAN Parameters

Device Setup Guides

Device Details / 0004A30B00215B73

Device 0004A30B00215B73

Name 00-04-A3-0B-00-21-5B-73

Description No description available

EUI 0004A30B00215B73 big endian (use by default)
735B21000BA30400 little endian (for LoRaWAN non-compliant devices)

AppEUI BE7A00000001109 big endian (use by default)
0911000000007ABE little endian (for LoRaWAN non-compliant devices)

AppKey you get from "LoRaWAN parameters" in the Device menu:

mnpe@fotonik.d... Logout

Back To Devices

DEVICE
00-04-A3-0B-00-21-5B-73

LoRaWAN Parameters

LoRaWAN AES128 Keys

AppKey
Application Key (Device Key)

... 6AFFC93B7B

Remove appkey

If you want to enable over-the-air join, add or derive the device's application

Only AppKey and AppEUI is needed for OTAA join procedure. For ABP (Activation by personalization) join procedure you need Device address (DevAddr) as well as AppSKey and NwkSKey. Note that these are security sensitive session keys. Therefore ABP is less secure than OTAA. Again, in this case we join the network using OTAA so we will not use the AppSKey and NwkSKey session keys. When using OTAA the sessions keys are generated on the device upon joining the network.

12. Now returning to the Arduino IDE. In the code we have the option of using either OTAA or ABP. You will need to un-comment the OTAA line and insert the AppEUI and AppKey you got from the LoraWAN server when adding the device.

```
//ABP: initABP(String addr, String AppSKey, String NwkSKey);  
//join_result = myLora.initABP("02017201", "8D7FFEF938589D95AAD928C2F8  
  
//OTAA: initOTAA(String AppEUI, String AppKey);  
join_result = myLora.initOTAA("70B3D57ED00001A6", "A23C96EE13804963F8
```

13. Deciding what to transmit. In the Arduino IDE, look for this code:

```
Serial.println("TXing");  
myLora.tx("!"); //one byte, blocking function
```

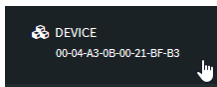
In this case we are transmitting the “!” character. You can change this into something else that you would like to send to the server.

14. In the loop function, change the delay between each transmission. It is set to 200ms which is too frequent. Change this to 2000ms.

```
void loop() {  
    led_on();  
  
    Serial.println("TXing");  
    myLora.tx("!"); //one byte, blocking function  
  
    led_off();  
    delay(200);
```

15. Now upload the code again, open the serial monitor, and this time you should see a message that the device has successfully joined the network, and finally that it is transmitting. It will transmit your message every 2 seconds.

Note: If you are unable to join the network it may be either a) wrong AppKey and AppEUI or b) You do not have Cibicom LoraWAN network coverage where you are. Try a different location. If possible close to DTU where there is coverage.

16. In the Lorient page, click the Device  to see an overview. Here you can see “Last radio” info:

Last radio	
Last seen	Mar 2, 2019, 7:08:54 PM
RSSI	-83.00 dBm
SNR	-21.00 dB
Device SNR	N/A
Frequency	868.300 MHz
SF	12
Bandwidth	125 kHz
Gateway	7076FFFFFF019F0F

And also “Last data” which is the payload we sent:

Last data (10 latest records)			
SeqNo	Time	Port	Data
5	a minute ago	1	0e

Congratulations, you have now transmitted data over a LoraWAN network!

17. The data seen above in “Last data” is only for your device. The device however is sharing application with other devices as well. The data received from all devices can be seen if you go back and click on **WebSocket Applications** in the menu and then on the WebSocket sample (see below):



WebSocket sample by LORIoT

Sample application that displays history of data in a table. You can send

<https://iotnet.teracom.dk/application/BE7A1296/websocket>

Click this link to listen to all the data sent to this Application. Notice that you will see data from other devices using the same Application. It will look something like this:

IoT course WebSocket

Connected

19 entries

Decode Data From Device

Send Data To Device

Device EUI	Local time	Freq [MHz]	Date rate	RSSI (dBm)	SNR (dB)	FCntUp	Port	Payload
0004A30B00F0FD7F	2021-03-18 22:00:52 .178	868.100	SF12 BW125 4/5	-95	-18	6	1	21
0004A30B00F0FD7F	2021-03-18 22:00:46 .746	867.100	SF12 BW125 4/5	-98	-14	5	1	21
0004A30B00F0FD7F	2021-03-18 22:00:41 .320	867.700	SF12 BW125 4/5	-97	-4	4	1	21
0004A30B00F0FD7F	2021-03-18 22:00:35 .918	867.500	SF12 BW125 4/5	-111	-21	3	1	21

Notice now you can see which spreading factor (SF), received signal strength (RSSI) and frequency that was used for the transmissions.

After some transmissions you may hit the duty-cycle limit and suddenly you see the ESP8266 attempting to transmit but nothing appears in the Cibicom backend. The

RN2483 is doing duty-accounting and denies to send the data until a channel is free. This can be bypassed by re-joining the network / turning on-off the device.

Time-on-air, spreading factor (SF) and data-rate

18. When transmitting with LoraWAN, the data be transmitted with either spreading factor SF7, SF8, SF9, SF10, SF11 or SF12. Increasing spreading factor means increased time-on-air, increased range and reduced data rate. Increased time-on-air means increased time until we're OK to transmit again, refer 1% duty cycle. The difference of spreading factor is considerable. Transmitting 8 bytes at SF7 takes 56ms and at SF12 it takes 1483ms for the same 8 bytes. Again, this has an impact on how long it takes until another transmission can be made because of the duty cycle restrictions. Time on air can be found using this tool: <https://www.thethingsnetwork.org/airtime-calculator>.

Furthermore, at SF12 the modem is on, spending 25mW (14 dBm) for much longer time. To reduce power consumption (= longer battery lifetime) a lower SF is preferred. To increase transmission range, a higher SF preferred. I.e. it is a trade-off between multiple factors.

The library used in this exercise is made for TTN, which per default does not allow for adaptive data rate (ADR). However Cibicom does. When ADR is enabled, the server can request the LoraWAN end-device (RN2483) to adapt SF to conditions (up or down). Referring to the RN2483 Commands reference user guide we find that the command "mac set adr on" will enable ADR in RN2483.

The rn2xx3.h library allows for sending raw commands to the RN2483 module using the function `sendRawCommand(F("<command>"))`. If our library instance is called `myLora` we can use `myLora.sendRawCommand(F("mac set adr on"))`; to enable ADR.

Include this command in the loop function, upload to the module, and inspect on the Cibicom backend if the SF changes from SF12. If not, the server has decided that SF 12 is the necessary spreading factor to be used.

19. To force the SF to change, we can use the command "mac set dr X" [X = 0, 1, 2, 3, 4 or 5] to force a different SF: dr (data rate) = 0 equals SF12 and dr = 5 equals SF7. Try to set a new data rate (e.g. "mac set dr 1") and observe the SF on Cibicom backend. If you manage to get the signal through at dr = 5 (SF7) you will be allowed to send more often before the duty-cycle limit kicks in.

"mac get dr" and "mac get adr" can be used to check the value of data-rate and ADR respectively. E.g.

String ADRstate = `myLora.sendRawCommand(F("mac get adr"))`;
will return *on* or *off* to the ADRstate variable.

Sending data downlink to the RN2483 from Cibicom/Loriot server

20. From the interface in 17 we can also send data to the RN2483. This can be done by clicking on “Send data to device” and copy past in the Device EUI you want to send data to, type some data and click send. Choose any port between 1 and 255.

IoT course Websocket

Connected 19 entries

Decode Data From Device Send Data To Device

Device EUI	Local time	Freq [MHz]	Date rate	RSSI (dBm)	SNR (dB)	FCntUp	Port	Payload
0004A30B00F0FD7F	2021-03-18 22:00:52.178	868.100	SF12 BW125 4/5	-95	-18	6	1	21

Notice how the data gets “queued for sending”. The data is not sent until the RN2483 device transmits and thereby opens up for the one the RX windows.

21. At this point however we have not prepared our device to receive the downlink message. To view the data sent to the device (RN2483) you need to tell the ESP8266 to fetch the received data from the RN2483. The default sketch code does not provide for this. To fetch the data, use the rn2xx3.h library function `getRx()`. If your instance is called `myLora`, use `str = myLora.getRx();` to load the received data into the string, `str`. Then print `str` to the serial monitor.

Add `str = myLora.getRx();` to your loop and print `str` to serial monitor. After uploading changes, send a HEX message downlink to the device using the “Send data to device” function above. Following a successful transmission you should see the received data in your serial monitor.

Send Data To Device

Device EUI (16 hex digits) Port (decimal number, 1 to 223)

0004A30B00F0FD7F 1 ☐ Request confirmation

Payload (hex string)

FF00FF

Send to device Keep Sending Data ☐

After clicking “Send to device” you will see the data getting “enqueued for sending” and when the server receives the next message from our device it will send it “enqueued data sent”.

Device EUI	Local time	Freq [MHz]	Date rate	RSSI (dBm)	SNR (dB)	FCntUp	Port	Payload
0004A30B00F0FD7F	2021-03-20 19:01:03.406							(enqueued data sent)
0004A30B00F0FD7F	2021-03-20 19:01:03.048	867.900	SF7 BW125 4/5	-105	-1	11	1	21
0004A30B00F0FD7F								FF00FF (enqueued for sending)

Checking the serial monitor you should see the incoming message received:

TXing

Received message: FF00FF

22. `getRx()` will retain the last received message so it will keep printing the same message over and over again. I.e. it could be beneficial to use a switch statement to only print newly received messages in the serial monitor. Try to implement a switch statement in the loop function like illustrated below:

```
switch(myLora.txCnf("!")) //return defined in TX_RETURN_TYPE (enum type).
{
    case TX_SUCCESS:
    {
        Serial.println("TX successful and acknowledged");
        break;
    }
    case TX_WITH_RX:
    {
        str = myLora.getRx(); //fetch incoming downlink message from server
        Serial.print("Received message: ");
        Serial.println(str);
    }
}
```

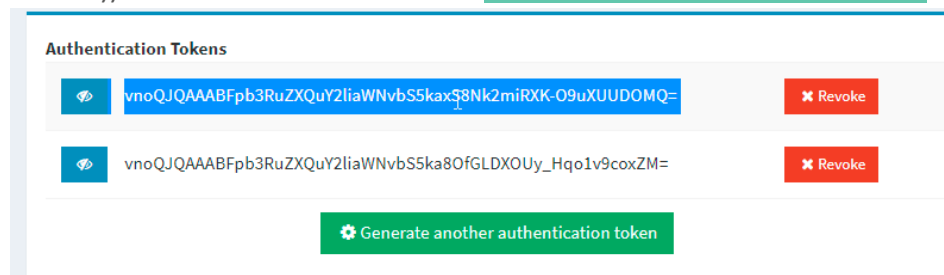
Notice that we here use the `txCnf()` function which will request a “received acknowledgement” or “ack” from the server. This will generate an extra downlink message as can be seen on the Cibicom backend.

▼ 0004A30B00F0FD7F	2021-03-20 18:58:22.965								(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:58:22.596	867.700	SF7 BW125 4/5	-107	3	2	1	21	(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:58:11.257								(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:58:10.902	867.300	SF7 BW125 4/5	-107	4	1	1	21	(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:57:58.378								(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:57:58.021	868.500	SF12 BW125 4/5	-86	-24	0	1	21	(enqueued data sent)
▼ 0004A30B00F0FD7F	2021-03-20 18:57:06.667								(enqueued data sent)

In the code example above we’re transmitting “!” (in HEX 0x21). You can transmit something else if you want.

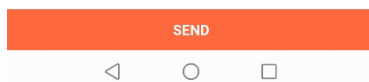
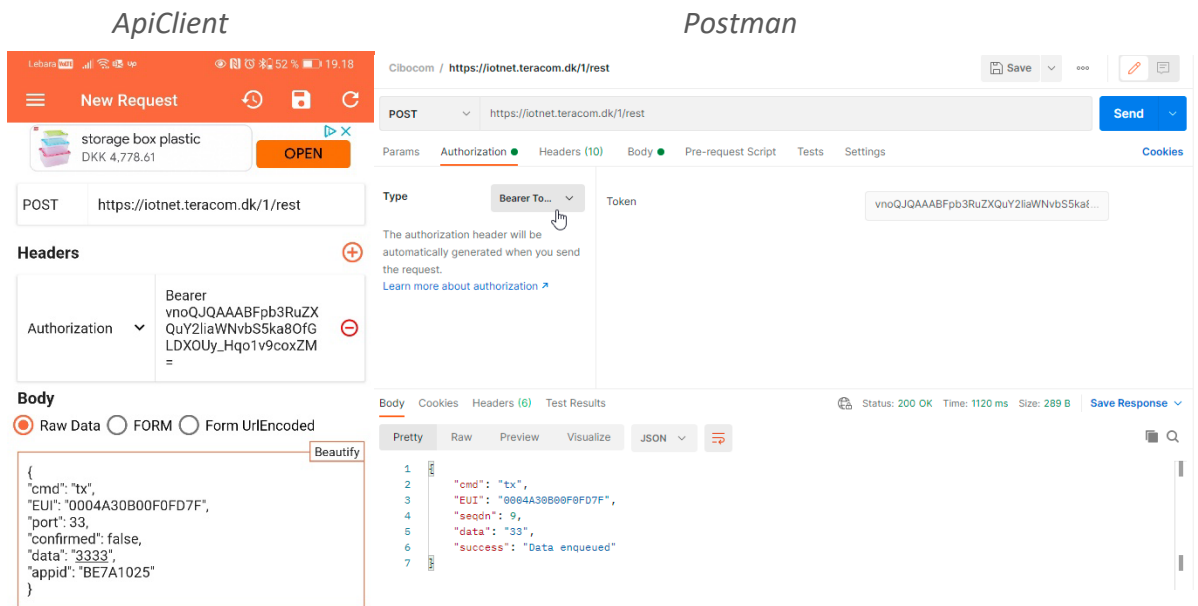
Sending data downlink to the RN2483 from 3rd party application

23. In practice it is clearly inconvenient to use the Cibicom / Lorient server backend to send downlink messages. The Cibicom / Lorient server has an API (using HTTP POST) that allows for posting downlink messages to the server from 3rd party applications. This can be done using various tools such as Postman (www.postman.com). The Lorient documentation (<https://docs.lorient.io/display/LNS/Send+Downlinks+via+API>) explains how to the API. Consult the documentation and collect the necessary elements (device EUI, App ID and Authorization token (APPLICATION ACCESS TOKEN)). The REST API URL to use is <https://iotnet.teracom.dk/1/rest>.

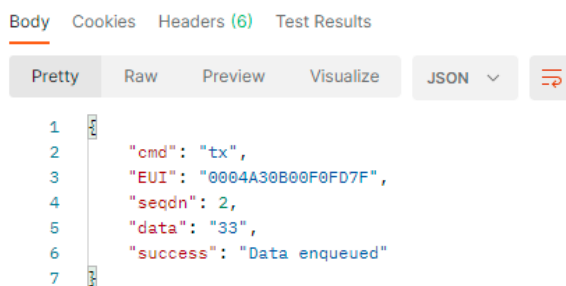


Authentication Token (choose either if more available). Click Access Token in Menu.

Using either Postman, a mobile app (e.g. *ApiClient*), Python or other, configure a downlink POST to the Cibicom server. See screenshots below for inspiration. Notice the “Bearer” prefix that is added to the Application access token.

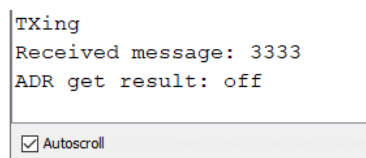


If you have configured it all correct and click Send you should get a response like below from the server (taken from Postman):



Upon inspecting the Cibicom backend you should see a “enqueued for sending” and/or “enqueued data sent”.

Finally, in the serial monitor we should find our received downlink message:

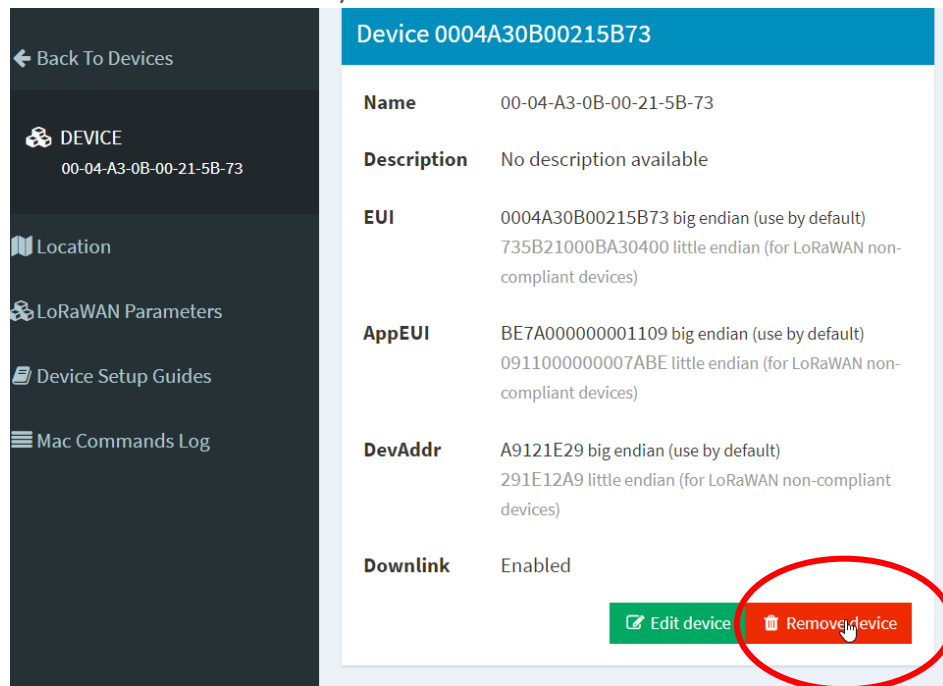


While postman or mobile apps such as *ApiClient* are not practical for a seamless implementation of LoraWAN downlink messaging, they illustrate the method that can as well be used with Python, Javascript, jQuery, Flutter and many other

development tools. I.e. using these tools and the method described above it is possible to develop a web or mobile application generating LoRaWAN downlink messages.

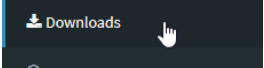
Final **Important** thing to do!

24. Final thing to do is to go and **remove your device** from the Device list. Otherwise the device is locked to this server and application and basically “bricked”. If you want to try out some of the final notes below, wait to remove the device, but please remember to do it when you are done.



Some final notes

- The Websocket application used here is for demonstration. There is no privacy and everyone with the right link can view the data.
- Application output can be chosen as needed. Azure, IBM Cloud, Google IoT, AllThingsTalk and more are available.
- Lots of information about each packet delivered to the server can be found and this can be useful. RSSI (received signal strength), SNR (signal to noise ratio), spreading factor, frequency, which gateways received the package (GW IDs), gateway time stamp, gateway locations (long, lat), sequence number and more. This data is extracted by the loriot server and is based on gateway/radio information derived from the package transmission. This formation can be retrieved through the

Application output. It can also be seen from the  section where you can fetch a .json file including this information about the past received packages.

json files

json

...R4r7ikM=

BE7A1109.json

Tip:

Open the .json in Firefox as this browser formats .json very nice and readable.

- Every time you power off and on the device (RN2483) you need to re-join the network

About duty cycle and RN2483, and how to by-pass it [only relevant with TTL/FTDI adapter available]

I'm getting a no_free_ch response from the module. How can I eliminate it?

This is entirely a duty cycle issue, which is regulatory. Generally, the duty cycle limit is 1% (but it does vary sometimes by frequency band and region) and RN2483 keeps "on air" and "off air" timers for each channel to help ensure that you don't violate the regulations. By default, there are three channels enabled, each with 0.33% duty cycle allocation. When pseudo-randomly hopping around the channels, the algorithm looks at the next channel and decides if the timers have enough remaining duty cycle allocation for the transmission. If not, then it skips that channel and moves to the next. If you've used all the allocation for all channels, you get the no_free_ch response. This is quite a common problem during development/learning/debugging, where you tend to send messages far more frequently than normal. In real "sleepy" applications it is rarely an issue. The no_free_ch means that all channels are busy transmitting and you have to wait until you can transmit again.

You can cheat the safeguards just by raising the duty cycle limit for each channel using the command `mac set ch dcycle` (from the "**RN2483 LoRa® Technology Module Command Reference User's Guide**", which could be found on the RN2483 product page under the "Documents" section) (<https://www.microchip.com/wwwproducts/en/RN2483>)

The value that needs to be configured can be obtained from the actual duty cycle X (in percentage) using the following formula: $= (100/X) - 1$

- `mac set ch dcycle 0 9` sets channel 0 to 10% (= 90% off)
- `mac set ch dcycle 1 99` sets channel 1 to 1% (= 99% off)
- `mac set ch dcycle 2 999` sets channel 2 to 0.1% (= 99.9% off).

Sending ACK/confirmed messages and receiving confirmations.

Send **un-confirmed**:

mac tx uncnf 1 [HEX Data]

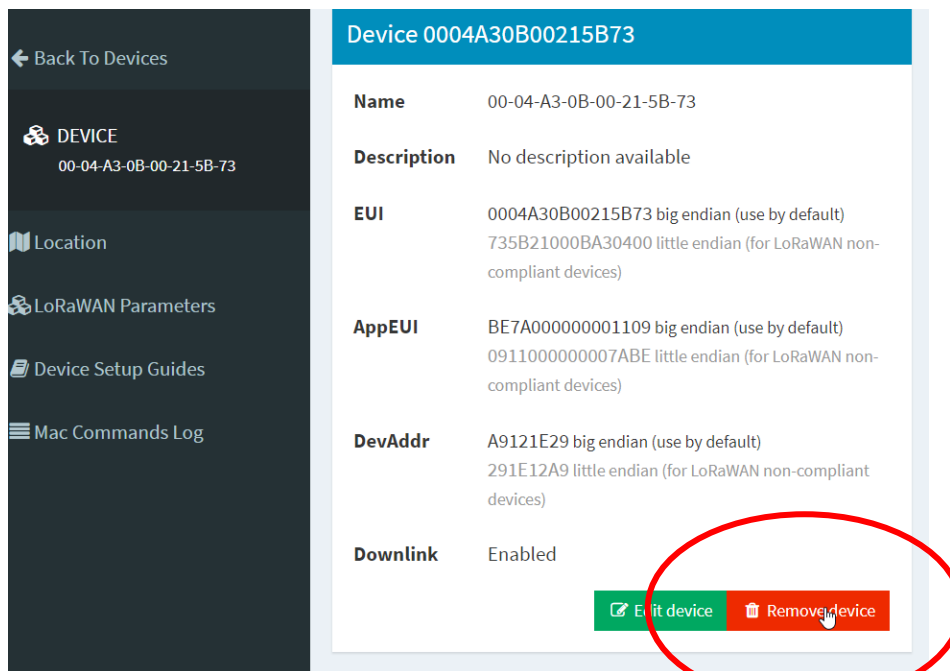
Answer to **un-confirmed** messages is always *mac_tx_ok*. This is not an ACK that they message has arrived at the server but just a notice that the message has been sent off.

Send **confirmed** messages (receive ACK that the message has arrived):

mac tx cnf 1 [HEX Data]

Answer to **confirmed** messages is *mac_tx_ok* (means message has arrived) or *mac_err* (message did not arrive, or ACK didn't get back to device). I.e. the answer (*mac_tx_ok*) really only makes sense when *mac tx cnf* is used.

Again, remember to go and remove your device from the Device list. Otherwise, the device is locked to this server and application and “bricked”.



On the device run the “*sys factoryRESET*” command to reset the device for the next user.