



Some USB-Serial converters feature a **+5v Tx** even when provided power is 3.3v !!! (CP2102 are ok)

# LoRaWAN end-devices user guide

Dr Thiebolt François

Modifications table

Date	Note
nov.19	added overview and general usage of our LoRaWAN platform
oct.19	update related to RN2483A FW 1.0.5 upgrade (class C end-devices)
aug.17	Initial release

## Abstract

This guide is both related to end-devices data exchanges with our lorawan infrastructure along with detailed information about hardware and software setup for some end-devices like the RN2483A lorawan module.

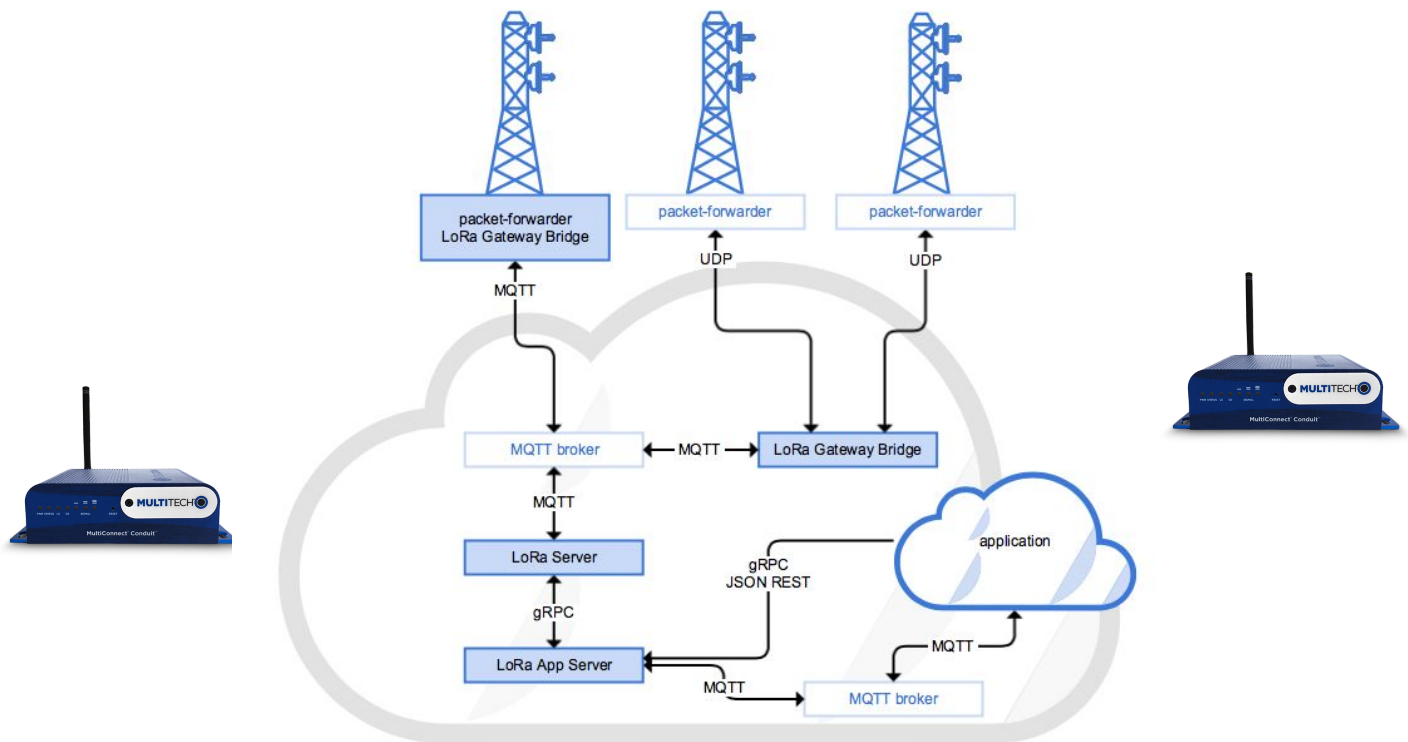
## Table of contents

Abstract	1
<b>Overview</b>	<b>3</b>
end-device data exchange	4
end-devices uplinks / downlinks	4
end-device identity	4
MQTT publish / subscribe	4
<b>My first data exchange</b>	<b>5</b>
<b>RN2483A + USB-serial converter</b>	<b>6</b>

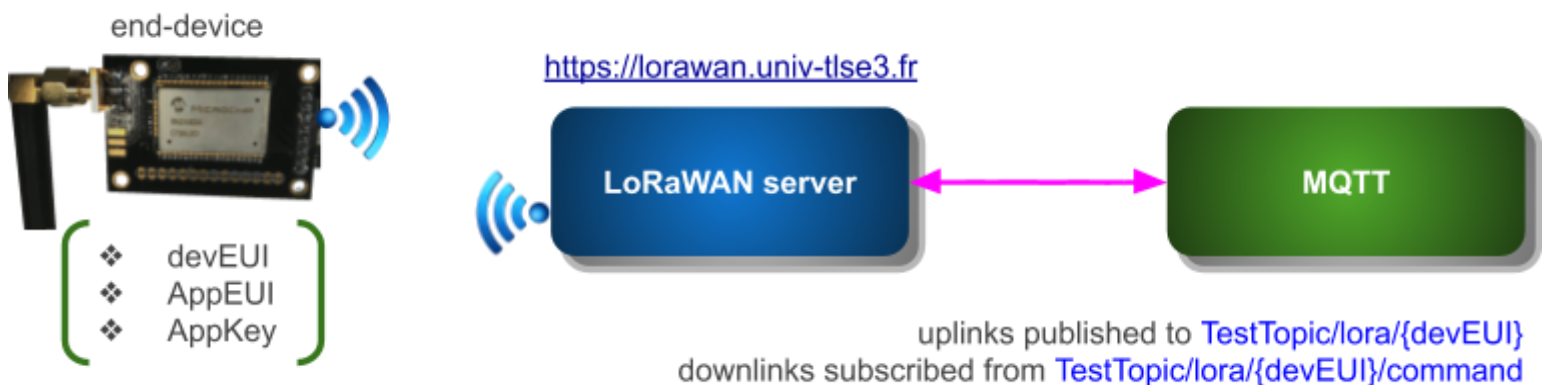
# Overview

neOCampus benefits from a federated LoRaWAN infrastructure built from several LoRaWAN gateways. These gateways does not process LoRa packets on their own but forward them to a central server: this is the packet forwarder mode.

In the end, it means that we're featuring a centralized end-devices management that will ease all of the lifecycle management process.



Overview of both uplinks and downlinks end-points at neOCampus MQTT broker:



## end-device data exchange

Before going on with details about data exchange, you first ought to read a bit of documentation to get accustomed with terms like:

- end-devices classes A, B or C,
- OTAA vs ABP,
- MQTT publish / subscribe paradigm.

In these practical exercises, we'll consider **A class end-devices** making use of the **OTAA** connexion procedure.

### end-devices uplinks / downlinks

Refers to the direction of data:

- **uplink** → end-device sending a data frame to the network,
- **downlink** → end-device receiving a data frame from the network.

### end-device identity

An end-device exhibit a unique **DevEUI**. It will get given access to an application specified through an **AppEUI** + credential specified as an **AppKey**:

- DevEUI
- AppEUI
- AppKey

These three parameters identify an end-device accessing an application.

At neOCampus, the application you'll be using is tied to the neOCampus MQTT broker.

### MQTT publish / subscribe

As specified earlier, whenever an end-device send data to the network, this data will get **published** by our LoRaWAN server to a topic in our MQTT broker:

- data uplink end-points **TestTopic/lora/{devEUI}**

It means that you, as an end-user, to be able to retrieve this data, you need to **subscribe** to this topic.

To then send data to your end-device, you'll need to publish a message to

- data downlink end-point: **TestTopic/lora/{devEUI}/command**

# My first data exchange

As a first step, you'll need to setup your end-device [RN2483A + USB-serial converter](#)

Then apply for the allocation of a DevEUI + Application credentials

You'll now write some **python3** code making use of *pyserial* python module to be able to connect (i.e 'join') with the gateway through the serial link of your RN2483A.

At this step, you can see the ongoing connexion operation through:

<https://lorawan.uni-tlse3.fr>

Now that you've been able to 'join' with the lorawan infrastructure, it's time to send a simple 'Hello world' and to retrieve it at the neOCampus MQTT broker. To achieve this in a simple way, I recommend to make use of already written applications like 'MQTTBox' (for example) that you'll find in the Google store app (free one of course)

To retrieve the data sent, you first need to connect with your MQTT client to the neOCampus MQTT broker:

The screenshot shows the MQTTBox application window with the title bar 'MQTTBox'. Inside, there's a 'Menu' button and a 'MQTT CLIENT SETTINGS' header. The settings are organized into a grid:

- MQTT Client Name:** An empty text input field.
- MQTT Client Id:** A text input field containing '0a725548-f9ff-4fd8-8349-6' with a refresh icon.
- Append timestamp to MQTT client id?:** A checked checkbox labeled 'Yes'.
- Broker is MQTT v3.1.1 compliant?:** A checked checkbox labeled 'Yes'.
- Protocol:** A dropdown menu showing 'mqtt / tcp'.
- Host:** An empty text input field.
- Clean Session?:** A checked checkbox labeled 'Yes'.
- Also connect on app launch?:** An unchecked checkbox labeled 'No'.
- Username:** An empty text input field.
- Password:** A text input field with masked characters '.....'.
- Reschedule Pings?:** A checked checkbox labeled 'Yes'.
- Queue outgoing QoS zero messages?:** A checked checkbox labeled 'Yes'.
- Reconnect Period (milliseconds):** A text input field containing '1000'.
- Connect Timeout (milliseconds):** A text input field containing '30000'.
- KeepAlive (seconds):** A text input field containing '60'.
- Will - Topic:** A text input field containing 'Will - Topic'.
- Will - QoS:** A dropdown menu showing '0 - Almost Once'.
- Will - Retain:** An unchecked checkbox labeled 'No'.
- Will - Payload:** An empty text input field.

You create a new connection with:

- MQTT client name → whatever you want
- MQTT client id → **don't touch** (ought to be unique!)
- Protocol: **mqtt/tcp**
- Host: **neocampus.univ-tlse3.fr**
- login / password: **test** / **<ask me>**

Once connection is established, subscribe to your topic :)

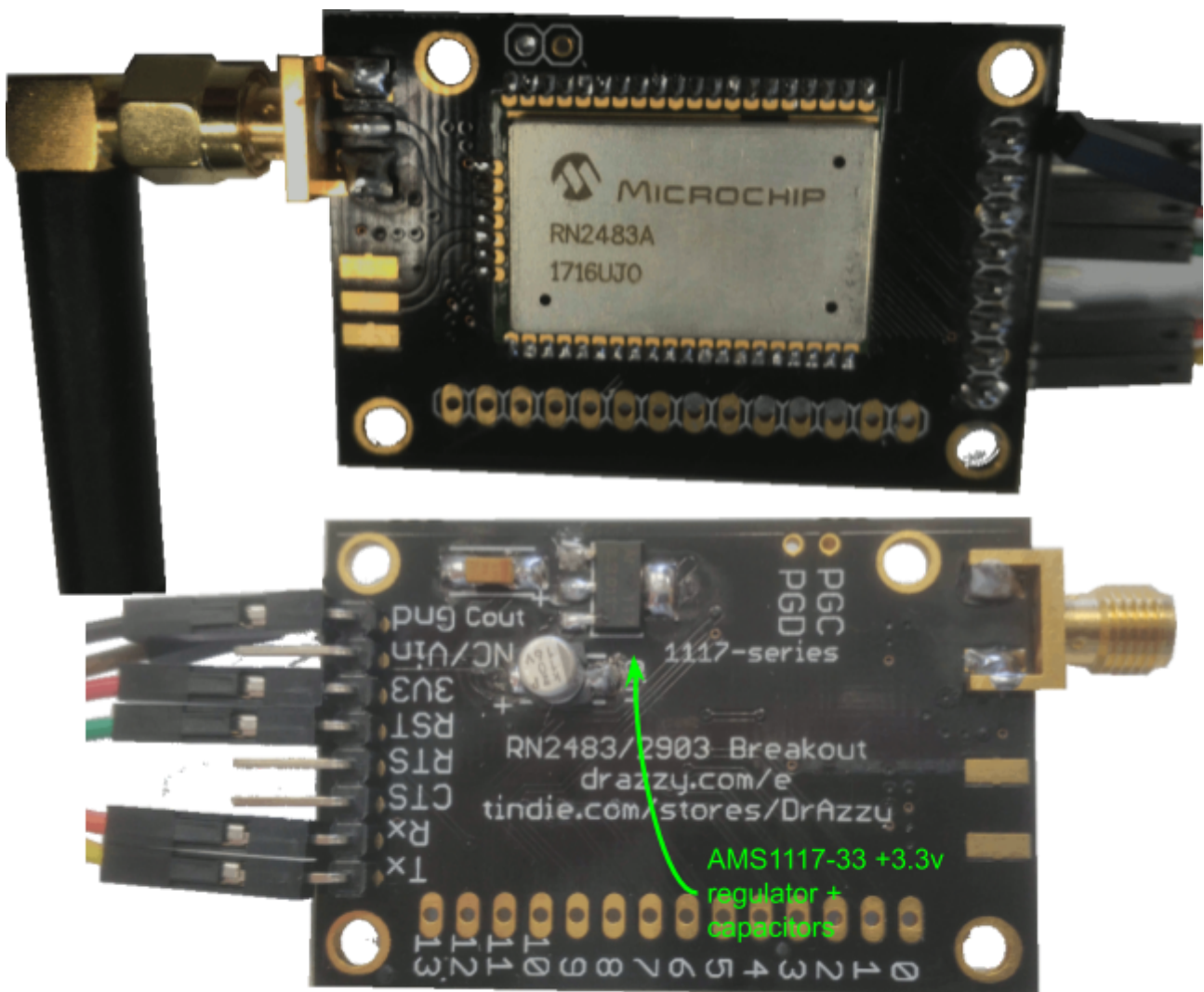
## RN2483A + USB-serial converter

One of the very first widely available end-devices, it is still fairly popular due to its simplicity.

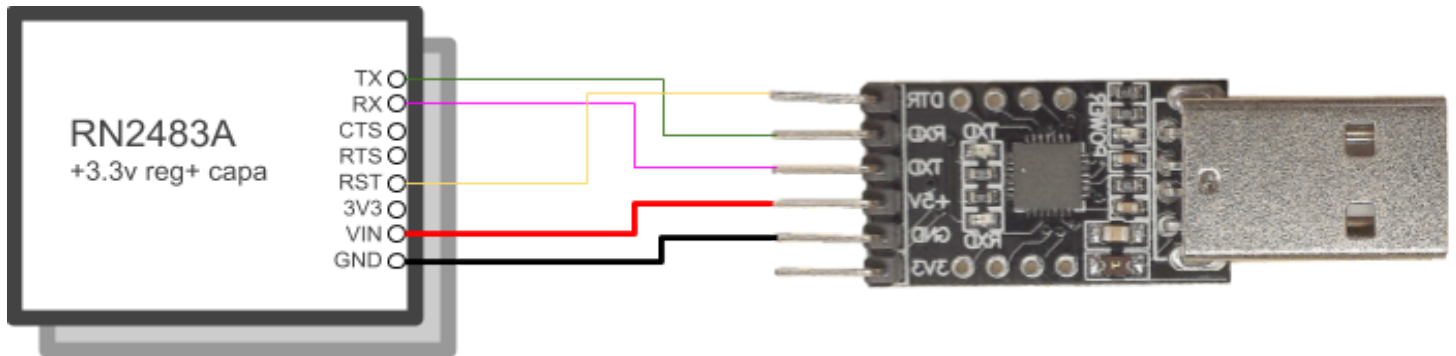
Microchip RN2483 is a LoRaWAN single chip solution that takes care of all of the LoRaWAN issues. Host microcontroller uses a **3.3v** serial link to communicate with.

In this scenario, we first start to setup a CP2102 based USB to serial converter along with a RN2483A on a dedicated breakout board.

In order to avoid power issues, we added a +3.3v regulator along with capacitors on the RN2483A breakout board.



For proper operation, it is best to reset RN2483A on serial link start. To do so, we'll make use of the CP2102's DTR line ... but this line is normally +VCC and then switch to GND once serial link is open ... thus we'll invert DTR polarity from software :)



Miniterm install

```
pip3 install pyserial
```

Launch terminal (in a tmux session ---better)

```
francois@smart[~] miniterm-3.py --dtr 0 /dev/ttyUSB0 57600
--- forcing DTR inactive
--- Miniterm on /dev/ttyUSB0 57600,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
RN2483 1.0.5 Oct 31 2018 15:06:52
```

*chip answer with its version number, here LoRaWAN stack revision 1.0.3 (i.e RN2483A)*

```
sys get ver
RN2483 1.0.5 Oct 31 2018 15:06:52
sys get hweui
0004A30B001EB4D1
```

*Note: commands won't appear because echo is OFF*