



neOCayenneLPP: LoRaWAN data exchange

Dr Thiebolt François, thiebolt@irit.fr

Specifications for the neOCampus / eCOnect LoRaWAN data exchange format.

Modifications table

Date	Note
jul.21	LPP_RAW type evolution + sample (Malik)
feb.21	reformat things
dec.20	Initial release by Julien Maignan ---ICAM Master2 internship @ IRIT

Abstract

LPWAN networks like LoRaWAN and SIGFOX are using the 868MHz ISM band. As a consequence, regulation rules apply and among them is the 1% duty-cycle. This means that if you're sending data that took 500ms to send, you won't be allowed to send a new packet before 49,5 seconds. Additionally, being almost over crowded, it's better to send short messages ... in a compact manner, hence the neOCayenneLPP format that specifies data exchange between LoRaWAN end-devices and our infrastructure.

Table of contents

Abstract	1
Overview	3
How much data could be sent in one shoot ?	3
ADR Adaptive Data Rate	3
neOCayenneLPP frame	4
preamble	4
tuple format	4
Frame format example	4
Data types	5
example: LPP_RAW	7
Data channels	8
positive values	8
negative values	8
Annexe-A Addendum	10

Overview

Sending data in a compact way while at the same time retaining a decent resolution is the challenge neOCayenneLPP is taking.

This format stems from CayenneLPP but it has been extended in many ways to be more compact and versatile. Especially, we're able to extend the kind of sensors (not limited to existing classes), having several numeric and analog sensors while being able to identify each of them.

Before we delve within the format specifications, pay attention to the fact that user payload has already side band data like:

- CRC
- end-device battery level

... thus no need to add them to your frame :)

How much data could be sent in one shoot ?

It is wise to stay below 200 bytes of payload. Moreover, the higher the number of bytes to send, the higher DataRate you ought to select ... meaning that you'll reduce your Spreading Factor thus being less error prone.

ADR | Adaptive Data Rate

In this mode, end-device and gateways define optimal conditions to communicate adjusting Data Rate and power level enhancing battery life.

neOCayenneLPP frame

neOCayenneLPP est une adaptation du format de données CayenneLPP (<https://github.com/ElectronicCats/CayenneLPP>). Il consiste à concaténer des données pour les transmettre en LoRa. Grâce à ce format, il est possible d'envoyer plusieurs données de différents types venant de plusieurs capteurs différents dans la même payload.

Une trame neOCayenneLPP se décompose de la façon suivante:

- **preamble**: 2 bytes → [format version] + [frame length]; note that frame length includes the preamble size
- **N x tuple** : a tuple is a group of bytes whose length depends on its own **type**

preamble

It is composed of a **version** byte followed by one byte giving the total **length** of the frame (including the preamble itself).

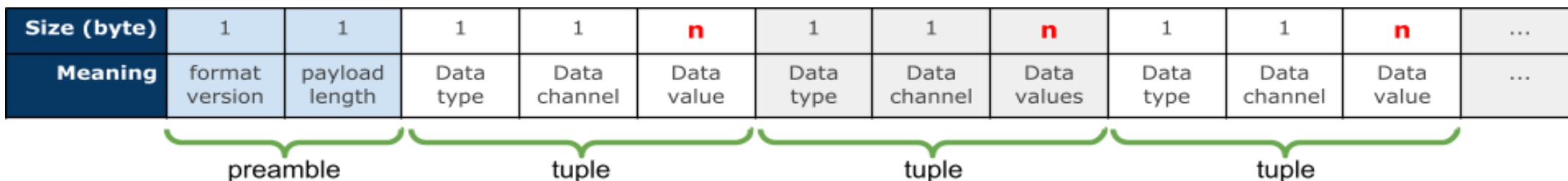
Version	Description
42	neOCampus LoRa v1
43	eCOnect LoRa v1

tuple format

Each tuple represents data the end-device would like to send back to the infrastructure. A **tuple** is composed of the following:

- **data type** : 1 byte; eg. LPP_LIMINOSITY, LPP_ANALOG_EXT; it represents the kind of data the tuple is hosting,
- **data channel** : 1 byte ble to uniquely identify a sensor; e.g I2C base address for I2C sensors
- **data value** : 1 to n bytes that represents the data itself. Length depends on the format as described later.

Frame format example



Data types

This is a **1** byte length field that describes the nature of the data from the current tuple but also specifies its encoding, example:

- LPP_LUMINOSITY → resolution is 1 lux, no offset, **data value** field is 2 bytes length, hence 0 to 65535 lux range
- LPP_TEMPERATURE → resolution is 0.25°C, 20° offset, **data value** field is 1 signed byte length, hence -11,75°C to 51,75°C range

Size is expressed as the number of bytes of the tuple's **data value** field.

Res. stands for *resolution*.


Type	ID	Size	Offset	Res.	Range	Notes
LPP_RAW	0	N				Intended to send raw data; WARNING : single tuple in whole frame (see example)
LPP_RAW_ANALOG	1	2				2 bytes raw analog value
LPP_RAW_ANALOG_EXT	2	4				4 bytes raw analog value
LPP_RAW_DIGITAL	3	1				1 byte raw digital value
LPP_LUMINOSITY	4	2		1 lux	[0 .. 65535 lux]	Lux value
LPP_PRESENCE	5	1		1	[0 .. 1]	a PIR sensor for example
LPP_FREQUENCY	6	2		1	[0 .. 65535]	e.g changing frequency acquisition of a sensor
LPP_TEMPERATURE	7	1s	20	0,25	[-11,75°C .. 51.75°C]	celsius unit, signed data value
LPP_HUMIDITY	8	1		0,5	[0 .. 100%]	%r.h unit
LPP_CO2	9	2		1	[0 .. 65535 ppm]	ppm unit
LPP_RAW_AIR_QUALITY	10	2				intended to send 'air quality' type raw values
LPP_RAW_GPS	11	9				TBC : GPS position

LPP_ENERGY	12	2		1	[0 .. 65535 WH]	Wh unit
LPP_ENERGY_SOLAR	13	2		1	[0 .. 65535 W/m2]	W/m2 (solar panel production)
LPP_ENERGY_WATT	14	2		1	[0 .. 65535 W]	Watt unit
LPP_ENERGY_VOLT	15	1s	230	0,1	[217,2 .. 242,7 v]	volt unit, signed data value
LPP_ENERGY_AMP	16	1		0,1	[0 .. 25,5 A]	Ampere unit
LPP_ENERGY_PHASE	17	2		1	[0 .. 65535 W]	Watt unit
LPP_UV	20	1		1e-2	[0 .. 2,55 mW/cm2]	TBC mW/cm2 (instantaneous power)
LPP_UV_ENERGY	21	1		1e-2	[0 .. 2,55 mJ/cm2]	TBC mJ/cm2 (energy ---joules)
LPP_WEIGHT	22	2		1	[0 .. 65535 g]	gramme unit
LPP_PRESSURE	23	1	990	1	[990 .. 1245 mBar]	mBar unit

LPP_RAW_XXXs need additional info for data types disambiguation; this can be achieved using the **channel** field of the tuple. This data won't get decoded by our LoRaWAN decoder but requires an external application able to extract its value(s).

- example: LPP_RAW

Size (byte)	1	1	1	1	n
Meaning	42	n +4	0	Data channel	Data values



Data channels

The 'Data channel' field is **1 signed** byte length and is part of a tuple intended to operate disambiguation about the identity of the sensor.

Since we have both numeric (I2C) and analog sensors, we decided to apply the following plan to the meaning of this field:

Value	Note
positive [0 .. 127]	I2C base address of sensor
negative [-128 .. -1]	ID of analog sensor specific to each end-device

positive values

Quite simple for I2C sensors as their base address are 7 bits length, thus we give the I2C base address of each sensor.

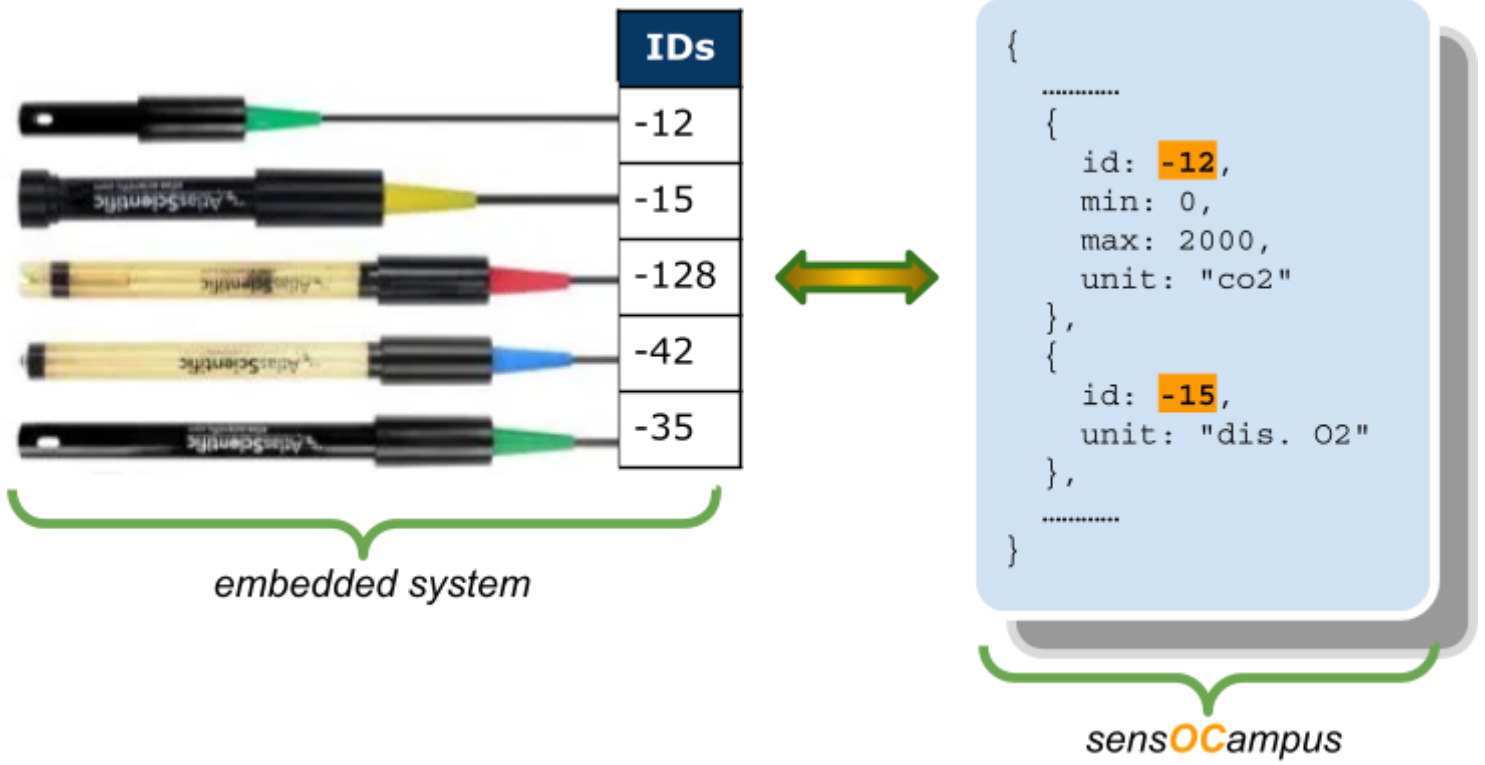
negative values

Analog sensors can't be determined by an end-device on itself ... hence we need some convention for each end-device. This way, the end-devices management application, senOCampus, will make a match between this negative ID and an analog sensor identity on a per-end-device basis.

For example, you could have an end-device embedding the following sensors:

- pH probe
- ORP probe
- dissolved oxygen probe
- ...

By means of 'sensOCampus', you will affect a **negative ID** along with a description of the sensor itself allowing the LoRaWAN decoder to extract data from frame and to send it directly to the database:



Annexe-A | Addendum