Fludia
Smart Energy Components

FM432 – User Guide

LoRaWAN IoT sensors for remote monitoring

Document Ref FLD10068 version 1.1.0

Product references:

FM432e: electricity meter optical reading
Ref: FM432e_nc_1mn, FM432e_nc_10mn, FM432e_nc_15mn

FM432ir: electricity infrared meter optical reading (SML protocol)
Ref: FM432ir_nc_1mn, FM432ir_nc_15mn

FM432g: gas meter optical reading (ATEX)
Ref: FM432g_nc_10mn, FM432g_nc_15mn

FM432p-a: pulse reading (ATEX)
Ref: FM432p-a_nc_1mn, FM432p-a_nc_10mn, FM432p-a_nc_15mn

FM432p-n: pulse reading (non-ATEX)
Ref: FM432p-n_nc_1mn, FM432p-n_nc_10mn, FM432p-n_nc_15mn

FM432t: temperature measurement
Ref: FM432t_1mn, FM432t_10mn, FM432t_15mn

Firmware versions

This documentation refers specifically to products including the following firmware versions. In most cases, data formats are the same for previous versions, but if something doesn't seem right, do not hesitate to contact support@fludia.com

FM432e

Optical head: FM210em_v5.4 (FM432e_nc_1mn), FM210em_v4.6 (FM432e_nc_10mn, FM432e_nc_15mn)
Radio/battery box: BPR07_V3.2.7

FM432ir

Optical head: FM210ir_v1.8 (FM432ir_nc_1mn), FM210ir_v1.6 (FM432ir_nc_15mn)
Radio/battery box: BPR07_V3.2.7

FM432g

Optical head: FM210g_v3.2
Radio/battery box: BPR07_v3.2.9

FM432p

Radio/battery box: BPR07_v3.3.1 (FM432p-a_nc_1mn, FM432p-n_nc_1mn), BPR07_v3.3.1 (FM432p-a_nc_10mn, FM432p-a_nc_15mn, FM432p-n_nc_10mn, FM432p-n_nc_15mn)

FM432t

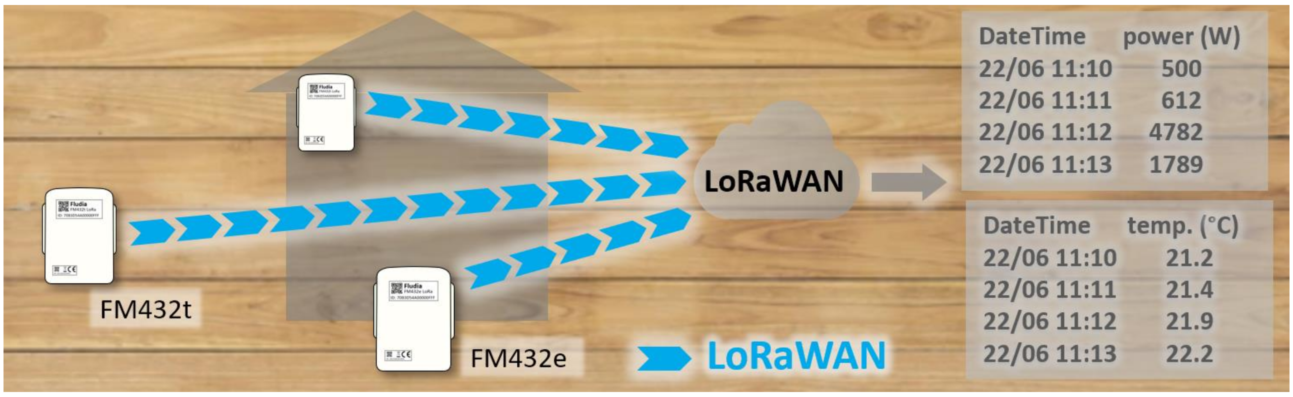
Radio/battery box: BPR07_v5.0.2

Revision history

Version	Notes	Date
1.0.1	New doc replacing ref FLD9492 (FM432e User Guide EN_v1.2.1), ref FLD9177 (FM432g User Guide EN_v1.2.1), ref FLD9899 (FM432p User Guide EN_v1.2.2), and a series of documents related to message decoding and data collect API	2022-05-16
1.0.2	Corrections related to T1 message headers; Additional code examples.	2022-05-25
1.0.3	Completion of FM432t sections	2022-05-30
1.0.4	Adding FM432ir code examples	2022-05-31
1.0.5	FM432 instead of FM232 in the illustration (overview chapter)	2022-06-01
1.0.6	Firmware version FM210ir_v1.6 replaces v1.2. Clarification (chapter 7.1): message contains 15 or 20 values depending on the type of meter.	2022-08-24
1.0.7	A Byte was missing in the T1 description for FM432ir (chapters 7.2 and 8.2)	2022-10-11
1.0.8	Adding JavaScript code examples for FM432e, FM432ir, FM432g, FM432p and FM432t.	
1.1.0	Correcting php code for FM432t. Technical messages headers. New FW versions for BPR07	2023-06-07

Table of Contents

1. Overview



The basic principle of this remote monitoring solution is to have IoT sensors (FM432x on the graph) using LoRa long range radio to transfer measurement messages to a LoRaWAN Network.

The **different types of sensors** available are:

- FM432e: optical reading of electricity meters (detecting rotating disk, or blinking light)
- FM432ir: optical reading of electricity meters **in Germany** (detecting rotating disk, or infrared port with SML protocol)
- FM432g: optical reading of gas meters (detecting digit rotation)
- FM432p: detecting pulse outputs of some meters (gas, water, electricity, heat…)
- FM432t: measuring temperature

The **installation process** consists of two main steps:

1. Installing the sensors: sticking the optical sensor to the existing meter, or connecting the wires, or just positioning the box, depending on the type of sensor.
2. Joining the LoRaWAN Network: starting up the sensor so that it sends join requests to the LoRaWAN Network (on which it must have been previously registered)

Retrieving the data can be done in one of three ways:

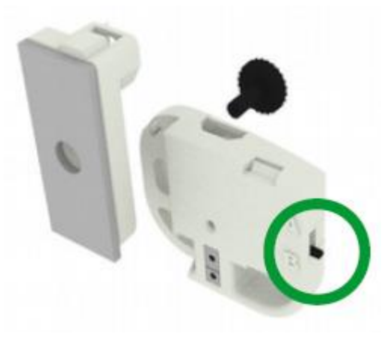
- Directly decoding the messages received by the LoRaWAN Network
- Connecting to the server's API https://fm430-api.fludia.com/v1/API/ (if a callback has been set up between the LoRaWAN Network and Fludia data server)
- Selecting data and downloading related files from the dashboard (if a callback has been set up between the LoRaWAN Network and Fludia data server)

2. Installing the sensors and joining the LoRaWAN Network

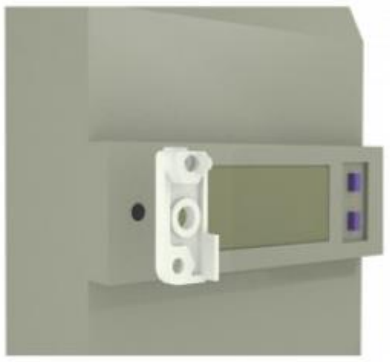
2.1 Installing the FM432e (optical reading of electricity meters)

First identify the type of meter you want to measure. It could be either an electronic electricity meter (with a blinking light) or an electromechanical electricity meter (with a rotating disk).

In case the **white cable** is plugged in the radio/battery box, it is recommended to **unplug it (**and leave only the optical side connected**) before proceeding** with the installation.

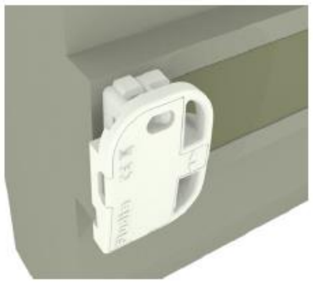


<u>If it is an **electronic meter**</u>, you must position the optical head **switch on B**, and if it is an **electromechanical meter**, you must position the **switch on A**:



If it is an **electronic meter**, stick the adhesive plastic mount on the meter in front of the meter blinking light (aim through the hole)

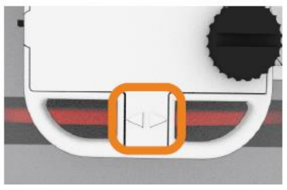
Then, position the optical head on top of the plastic mount and tight the black screw.



<u>If it is an **electromechanical meter**</u>, make sure the optical head is properly attached to the adhesive plastic mount with the help of the black screw:

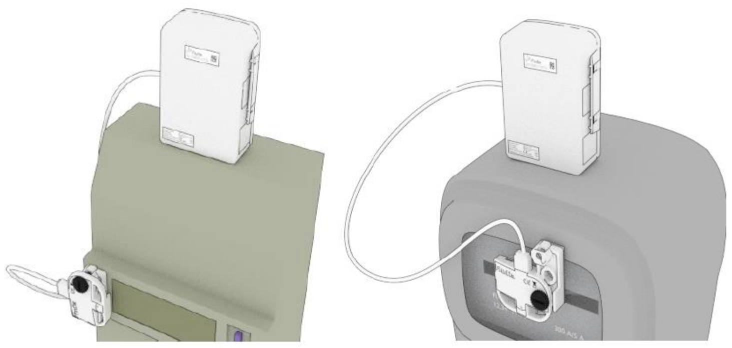
Stick the system on the meter glass panel, making sure the two arrows are perfectly aligned with the meter disk (face the meter and keep your eyes at disc level for better result):

If the arrows are not completely lined up with the disc, loosen the screw, adjust position, and tighten the screw



Connect the white cable between the optical head and the FM432 battery/radio box.

In the case of an electromechanical meter, make sure the optical head is still perfectly aligned with the disk.



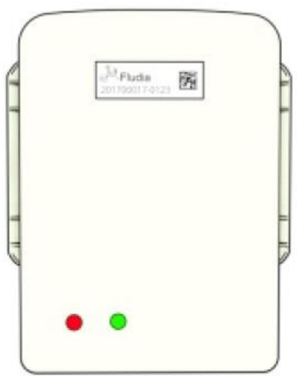
The FM432e starts measuring automatically.

The optical head LEDs (lights) should blink this way:

1. Calibration: red LED blinks for 20 seconds.
2. Validation: green LED blinks every time the meter light blinks (electronic meter) or every time the disk mark (black or red) comes in front of the sensor (electromechanical meter).
3. After 3 minutes, the green light stops completely to avoid wasting battery load.

The LEDs on the radio/battery box show the progress of **the join process**:

- Red and Green LED blinking together (join in progress)
- Green LED blinking alone (join successful)
- Red LED blinking alone (join failed)



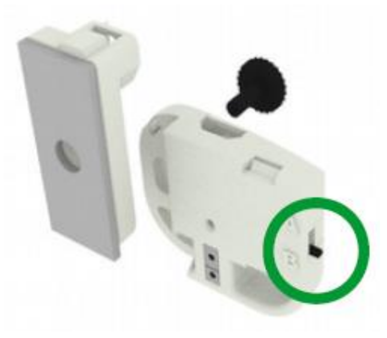
If the LEDs on the radio/battery side blink green twice, red once, green twice, red once and so on, it means that the sensor has previously joined a LoRaWAN Network (but not necessarily the one you want your sensor to join now). If you want to make sure, remove the batteries, wait for 10 seconds, and put them back on. It resets the join sequence so that your sensor can join from a fresh start.

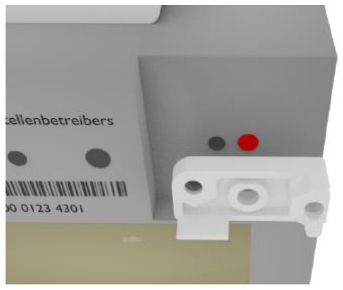
2.2 Installing the FM432ir (optical reading of electricity meters in Germany)

First identify the type of meter you want to measure. It could be either an mME electricity meter (with an infrared port supporting SML protocol) or an electromechanical electricity meter (with a rotating disk).

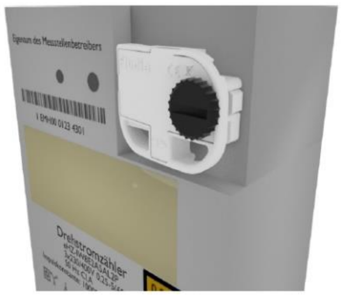
In case the **white cable** is plugged in the radio/battery box, it is recommended to **unplug it (**and leave only the optical side connected**) before proceeding** with the installation.

If it is an **mME meter**, you must position the optical head **switch on B**, and if it is an **electromechanical meter**, you must position the **switch on A**:





<u>If it is an **mME meter**</u>, stick the adhesive plastic mount on the meter in front of the meter infrared receiving LED (aim through the hole)

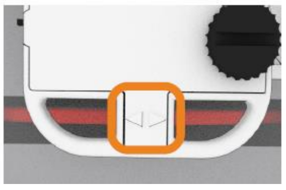


Then, fasten the optical head to the plastic mount.
Use the black screw to tighten the optical head to the mount.

<u>If it is an **electromechanical meter**</u>, make sure the optical head is properly attached to the adhesive plastic mount with the help of the black screw:

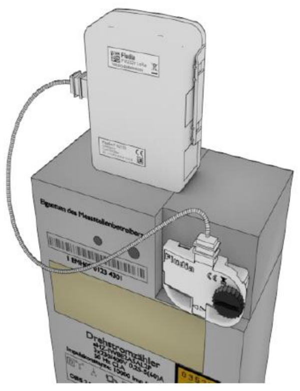
Stick the system on the meter glass panel, making sure the two arrows are perfectly aligned with the meter disk (face the meter and keep your eyes at disc level for better result):

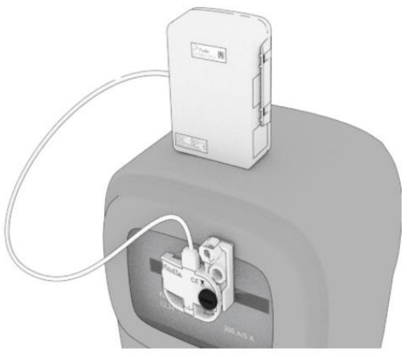
If the arrows are not completely lined up with the disc, loosen the screw, adjust position, and tighten the screw.



Connect the white cable between the optical head and the FM432 battery/radio box.

In the case of an electromechanical meter, make sure the optical head is still perfectly aligned with the disk.

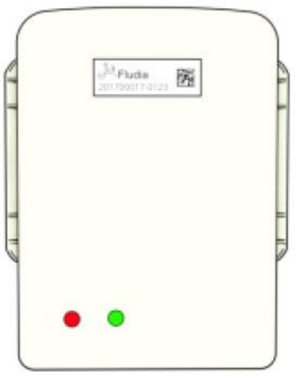




The FM432ir starts measuring automatically.

The **optical head LEDs** (lights) should blink this way:

1. Calibration: red LED blinks for 20 seconds.
2. Validation: green LED each time the disk mark (black or red) comes in front of the sensor (electromechanical meter) or each time a correct infrared signal is detected (mME meter).
3. The green light stops completely to avoid wasting battery load (after 3 minutes for electromechanical, after 1 minute for mME).

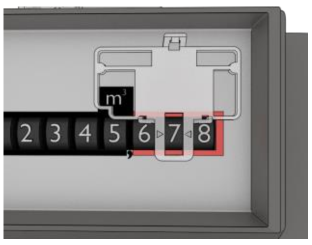


The LEDs on the radio/battery box show the progress of **the join process**:

- Red and Green LED blinking together (join in progress)
- Green LED blinking alone (join successful)
- Red LED blinking alone (join failed)

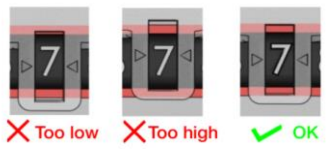
If the LEDs on the radio/battery side blink green twice, red once, green twice, red once and so on, it means that the sensor has previously joined a LoRaWAN Network (but not necessarily the one you want your sensor to join now). If you want to make sure, remove the batteries, wait for 10 seconds, and put them back on. It resets the join sequence so that your sensor can join from a fresh start.

2.3 Installing the FM432g (optical reading of gas meters)

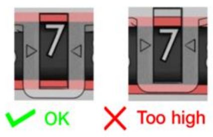


Clean the meter window.

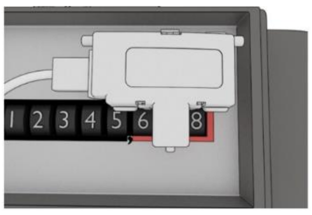
Paste the holder on the glass of the meter, in front of the **prior-to-last digit** of the index, using the provided adhesive (already fixed on the holder).



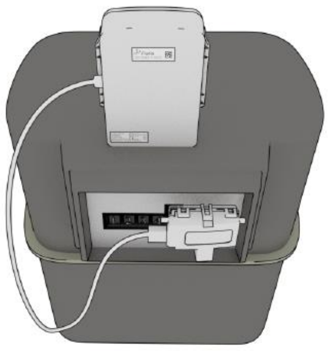
>> Make sure the arrows of the holder are perfectly at **mid-height of the frame**.



>> In some cases, the digit can be higher or lower than normal. Position the arrows at **mid-height of the frame**, **not** at mid-height of the **digit.**



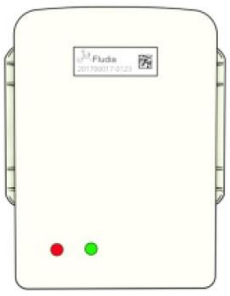
Clip the optical sensor to the holder.



Connect the white cable between the optical head and the FM432 battery/radio box.

The FM432g starts measuring automatically.

The optical head red LED (light) should blink for 20 seconds.



The **LEDs on the radio/battery box** show the progress of **the join process**:

- Red and Green LED blinking together (join in progress)
- Green LED blinking alone (join successful)
- Red LED blinking alone (join failed)

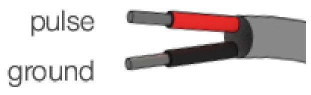
If the LEDs on the radio/battery side blink green twice, red once, green twice, red once and so on, it means that the sensor has previously joined a LoRaWAN Network (but not necessarily the one you want your sensor to join now). If you want to make sure, remove the batteries, wait for 10 seconds, and put them back on. It resets the join sequence so that your sensor can join from a fresh start.

2.4 Installing the FM432p (detecting meter pulse outputs: gas, water, elec, heat…)

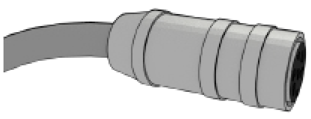
The FM432p comes with a cable that needs to be connected to the meter pulse interface.

By default, the end of the cable shows two wires: red wire (pulse), black wire (ground).

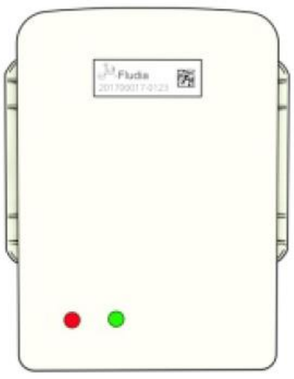
If the meter output shows a polarity, make sure to connect the cable accordingly.



If the cable is equipped with a “binder” connector, just connect it to the pulse output interface. The “binder” plug can be wired either in 3-5 configuration (default) or in 4- configuration (then, there is a colored ring around the cable).


"binder" connector

The FM432p starts measuring automatically once it has detected a first pulse.



The **LEDs on the radio/battery box** show the progress of **the join process**:

- Red and Green LED blinking together (join in progress)
- Green LED blinking alone (join successful)
- Red LED blinking alone (join failed)

If the LEDs on the radio/battery side blink green twice, red once, green twice, red once and so on, it means that the sensor has previously joined a LoRaWAN Network (but not necessarily the one you want your sensor to join now). If you want to make sure, remove the batteries, wait for 10 seconds, and put them back on. It resets the join sequence so that your sensor can join from a fresh start.

2.5 Installing the FM432t (measuring temperature)

Start the join process by removing the batteries, waiting for 10 seconds, and putting them back on.

The **LEDs on the radio/battery box** show the progress of **the join process**:

- Red and Green LED blinking together (join in progress)
- Green LED blinking alone (join successful)
- Red LED blinking alone (join failed)

Position the FM432t box wherever you want to measure temperature. For measuring in-house temperature, it is recommended to position the FM432t box around 1 meter above the floor and, if possible, not too close to a wall (especially outside walls).

Warning: the FM432t box is not “rain-proof”. So, to measure outside temperatures, it is recommended either to position it in a place protected from the rain, or inside a “rain-proof” additional box (a full waterproof box should be avoided because of the risk of water vapor infiltration and condensation).

3. Replacing batteries in a sensor

Batteries used in FM432x products are special Lithium 3.6V batteries (Lithium thionyl chloride or Li-SoCl2). You should only replace them by similar batteries, of good quality, form renown manufacturers, and make sure that they are positioned the right way (+ side and – side are indicated both on each battery and at the bottom of the battery compartment).

The usual format of the batteries is A, but AA works also (though the energy is lower).

It is possible to use only one battery instead of two, but the expected lifetime will be of course accordingly reduced.

In case of an ATEX certified product such as the FM432g, when operating in an ATEX zone you should only use certified batteries provided by Fludia, so that the product is still considered certified.

4. Removing the batteries to make a sensor forget its join

To make a sensor forget its join, you simply must cut the energy source: remove the batteries.

5. Decoding FM432e_nc_1mn messages

5.1 Types of messages

	Time-step
Message Type	**1 minute**
Data message (T1)	Every 20 minutes (20 x 1 minute)
Service message (T2)	Every 24 hours

FM432e_nc_1mn generates two kinds of messages:

- T1: contains twenty power values and the last index. It is generated every 20 minutes.
- T2: contains additional information such as the sensor type and the firmware version. It is generated once per day.

5.2 Data message (T1) structure

5.2.1 Introduction

The T1 data message transmits 1 **index** value (cumulated energy value) and 20 **power** values.

A power value is the average power over 1 minute. The average power calculation is based on the elapsed time between two consecutive optical detections and involves interpolation in order to create 1 minute values. One optical detection can be one LED flash (in the case of an electronic electricity meter) or one disk rotation (in the case of an electromechanical electricity meter). In simple cases, 1 detection represents 1Wh (Watt-hour) and the power values are in W (Watts). In other cases, a multiplication factor x is related to the meter and 1 detection can represent xWh (in this case the power values should be multiplied by this factor once received).

The T1 data message behaves as follows:

1. First message is sent 30 minutes after the sensor start-up.
2. Subsequent messages are sent periodically every 20 minutes.

Every T1 data message includes data related to a period of 20 minutes.

5.2.2 Timestamping

Index value should be time stamped as follows: $t = t_{received} - delay$

Each **power** should be time stamped as follows: $t_i = t_{received} - delay - (20 - i)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,19}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *delay is a delay of 10 minutes due to the power computation mechanism. It means that the message is sent 10 minutes after the time of the last 1 minute power. For example, if the message contains power values between 9:00am and 9:20am, the message will only be sent at 9:30am. This way, even*

if energy is low, there is time to wait for a late optical detection, necessary to compute average power, and attribute their share to the power values before sending the message.

5.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	...	#40	#41	#42	#43	#44	#45
Signification	Header*	Index				P(t0)		P(t1)		P(t2)		...	P(t17)		P(t18)		P(t19)	

5.2.4 Header

*** Header value (Hexa)**	**Signification**
5b	FM432e with 1-min time step

5.2.5 Index calculation

The index can be obtained this way:

Index = (**Byte_2** * 2^24) + (**Byte_3** * 2^16) + (**Byte_4** * 2^8) + **Byte_5**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

5.2.6 Power values calculation

Power values are labelled (t0) to (t19).

The formulas for obtaining power values are:

Power(t0) = (Byte_6 * 2^8) + Byte_7
Power(t1) = (Byte_8 * 2^8) + Byte_9
Power(t2) = (Byte_10 * 2^8) + Byte_11
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

5.2.7 Example

Given the following T1 message:

5b0afdff00068f068f0649066a067e0682057a04ad049f04bd04c204c004c604bf04ae04a504a304b0049b04ac

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**...**	**#40**	**#41**	**#42**	**#43**	**#44**	**#45**
Signification	**Header***	**Index**				**P(t0)**		**P(t1)**		**P(t2)**		**...**	**P(t17)**		**P(t18)**		**P(t19)**	
Hexa value	5b	0a	fd	ff	00	06	8f	06	8f	06	49		04	b0	04	9b	04	ac
Decoded value		184418048				1679		1679		1609			1200		1179		1196	

5.3 Service message (T2) structure

The T2 data message transmits technical information about the sensor (firmware, meter type, battery status indicator) and more info about the measurements (a longer index, the time-step, and a max power value). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Signification	Header (51)	Number of starts	Time synchronization (see below)	Not used	Optical head information (see below)	Index				Not used		Time step (see below)

Number of starts

Byte	**Signification**
#2	Number of starts since initial configuration (= number of times the sensor is unplugged from the radio module)

Time synchronization information

Byte	**Bit**	**Signification**
#3	1 - 7	Jitter (in seconds) set to zero for 1 minute data
	8	Synchro querying: not available for 1 minute data

Optical head information

Byte	**Bit**	**Signification**
#5	1 - 6	Firmware version
	7	Meter type: 0 = electromechanical meter 1 = electronic meter
	8	Battery status: 0 = battery OK 1 = battery NOK (Battery voltage < 2.8 Volts)

Index

Bytes	**Index calculation**
#6, #7, #8, #9	**Index** = (**Byte_#6** * 2^24) + (**Byte_#7** * 2^16) + (**Byte_#8** * 2^8) + **Byte_#9**

Time-step

Byte	**Value (hexa)**	**Signification**
#12	02	1-minute interval

5.4 Code examples

We present examples of codes that perform the calculation for the index and for the power values.

5.4.1 *Python code example*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def decode_index(payload):
 if len(payload) == 90:
  index = int(payload[2:10],16)
  return index
 else:
  return None
# return power list from T1 payload
def decode_power_list(payload):
 if len(payload) == 90:
  power_list_hex = [payload[(10 + 2 * i):((10 + 2) + 2 * i)] for i in range(40)]
  power_list = [int(power_list_hex[i * 2]+power_list_hex[i * 2 + 1], 16) for i in
range(len(power_list_hex)/2)]
  return power_list
 else:
  return None
# Exemple :
payload =
"5b0afdff00068f068f0649066a067e0682057a04ad049f04bd04c204c004c604bf04ae04a504a304b0049b04ac"
index = decode_index(payload)
print "Index: " + str(index)
power_list = decode_power_list(payload)
if power_list is not None and index is not None:
 print "Power list: " + str(power_list)
else:
 print("the payload has the wrong size")
```

5.4.2 *PHP Code example*

```
<?php
// return index from T1 payload
function decode_index($payload)
{
 $index = hexdec(substr($payload,2,8));
 return $index;
}
function payload_to_hex($payload)
{
 if(strlen($payload) == 90)
 {
  for ($i = 0; $i < 40; $i++)
  {
   $power_list_hex[$i] = substr($payload,10 + 2 * $i,2);
  }
 }
 return $power_list_hex;
}
// return power list from T1 payload
function decode_power_list($payload)
{
 if(strlen($payload) == 90)
 {
  $power_list_hex = payload_to_hex($payload);
  $power_list = [];
  for ($i = 0; $i < 20; $i++)
  {
   $power_list[] = hexdec($power_list_hex[$i * 2].$power_list_hex[$i * 2 + 1]);
  }
  return $power_list;
 }
 else
 {
  return null;
 }
}
// Exemple :
$payload =
"5b0afdff00068f068f0649066a067e0682057a04ad049f04bd04c204c004c604bf04ae04a504a304b0049b04ac";
$index = decode_index($payload);
$power_list = decode_power_list($payload);
if ($power_list != null && $index != null)
{
 echo "Index: ".$index."\n";
 foreach ($power_list as $power)
 {
  echo $power. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

5.4.3 JavaScript Code example

```
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    if (payload.length == 90) {
        index = payload.substring(2, 8);
    }
    return parseInt(index, 16);
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var power_list_hex = []
    if (payload.length == 90){
        for(i=0;i<40;i++){
            power_list_hex.push(payload.substring((10+2*i),2+(10+2*i)))
        }
    }
    return power_list_hex
}

//return power list from T1 payload
function decode_power_list(payload) {
    var power_list = []
    var power_list_hex = []
    if (payload.length == 90){
        power_list_hex = payload_to_hex(payload)
    }
    if(power_list_hex.length == 40){
        for(i=0;i<20;i++){
            power_list.push(parseInt(power_list_hex[i*2]+power_list_hex[i*2+1], 16))
        }
    }
    return power_list
}
//Exemple
payload =
"5b0afdff00068f068f0649066a067e0682057a04ad049f04bd04c204c004c604bf04ae04a504a304b0049b04ac"
var index = null
var power_list = []
index = decode_index(payload)
power_list = decode_power_list(payload)
if(index && power_list.length == 20){
    console.log("Index: "+index)
    console.log("List of power values in W: "+power_list)
}
else{
    console.log("The payload has the wrong size")
}
```

6. Decoding FM432e_nc_10mn and FM432e_nc_15mn messages

6.1 Types of messages

Sensor version	10 minutes	15 minutes	1 hour
Data message (T1)	Every 80 minutes (8 x 10 minutes)	Every 2 hours (8 x 15 minutes)	Every 8 hours (8 x 1 hour)
Service message (T2)	Every 24 hours	Every 24 hours	Every 24 hours
Technical message 1 (TT1)	Every 24 hours	Every 24 hours	Every 24 hours
Technical message 2 (TT2)	Every hour for 1 day	Every hour for 1 day	Every hour for 1 day

FM432e generates four kinds of messages:

- T1: contains the consumption data and is generated several times per day with a frequency determined by the sensor time-step.
- T2: contains useful information such as the sensor type, the firmware version, a long index. It is generated once per day.
- TT1: contains technical information (for technical feedback purposes) and is generated once per day.
- TT2: contains technical information (for technical feedback purposes) and is generated each hour for one day after startup.

6.2 Data message (T1) structure

6.2.1 Introduction

The T1 data message transmits **1 index value** and **8 increments**.

The **index value** is the cumulated number of optical detections since the start of the sensor. One optical detection can be one LED flash (in the case of an electronic electricity meter) or one disk rotation (in the case of an electromechanical electricity meter). In simple cases, 1 detection represents 1Wh. In other cases, a multiplication factor is related to the meter and 1 detection can represent xWh.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent:

- every 80 minutes for 10-minutes time-step sensors
- every 2 hours for 15-minutes time-step sensors
- every 8 hours for 1-hour time-step sensors

6.2.2 *Timestamping*

Index value should be time stamped as follows: $t=t_{received}$

Each **increment** should be time stamped as follows: $t_i=t_{received} - ((8 - i)*Time\_step)$

Where:

- t_i is the timestamp for one element of the data message $i \in \{0,7\}$.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements.

6.2.3 *Byte structure*

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20
Signification	Header*	Index			Incr. (t0)		Incr. (t1)		Incr. (t2)		Incr. (t3)		Incr. (t4)		Incr. (t5)		Incr. (t6)		Incr. (t7)	

6.2.4 *Header*

*** Header values (Hexa)**	**Signification**
20	10-min time step
21	15-min time step
22	1-hour time step

6.2.5 *Index calculation*

The index can be obtained this way:

Index = (**Byte_2** * 2^16) + (**Byte_3** * 2^8) + **Byte_4**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

6.2.6 *Increment calculation*

Increment values are labelled (t0) to (t7) and represent successive increments.

The formulas for obtaining increment values are:

Increment(t0) = (Byte_5 * 2^8) + Byte_6
Increment(t1) = (Byte_7 * 2^8) + Byte_8
Increment(t2) = (Byte_9 * 2^8) + Byte_10
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

6.2.7 *Example*

Given the following T1 message: 21006f920178017b0181018c01980196019c019f

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**
Signification	**Header***	**Index**			**Incr. (t0)**		**Incr. (t1)**		**Incr. (t2)**		**Incr. (t3)**		**Incr. (t4)**		**Incr. (t5)**		**Incr. (t6)**		**Incr. (t7)**	
Hexa value	21	00	6F	92	01	78	01	7B	01	81	01	8C	01	98	01	96	01	9C	01	9F
Decoded value		28562			376		379		385		396		408		406		412		415	

6.3 Service message (T2) structure

The T2 data message transmits technical information about the sensor (firmware, meter type, battery status indicator) and more info about the measurements (a longer index, the time-step, and a max power value). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Signification	Header (0E)	Number of startups	Time synchronization (see below)	Param. Id	Sensor information (see below)	Long index				Max power value (reset after every T2 message)		Time step (see below)

Time synchronization information

Byte	**Bit**	**Signification**
#3	1 - 7	Jitter (in seconds)
	8	Synchro querying: 0 = no synchro querying 1 = synchro querying

Sensor information

Byte	**Bit**	**Signification**
#5	1 - 6	Firmware version
	7	Meter type: 0 = electromechanical meter 1 = electronic meter
	8	Battery status: 0 = battery OK 1 = battery NOK (Battery voltage < 2.8 Volts)

Time step

Byte	**Value**	**Signification**
#12	00	10-minutes time step
	03	15-minutes time step
	01	1-hour time step

6.3.1 Long Index calculation

The index can be obtained this way (make sure to first convert to decimal):

Index = (**Byte_6** * 2^24) + (**Byte_7** * 2^16) + (**Byte_8** * 2^8) + **Byte_9**

6.3.2 Max power value calculation

The max power in W can be obtained this way (make sure to first convert to decimal):

Pmax = (Byte_10 * 2^8) + Byte_11

6.4 Technical message #1 (TT1) structure

Header (hexa): 12

Size: 19 Bytes

6.5 Technical message #2 (TT2) structure

Header (hexa): 13

Size: 11 Bytes

6.6 Code examples

We present examples of codes that perform the calculation for the index and for the increments.

At the end of each code a way of converting increments in average power values is included.

6.6.1 *Python code example*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def decode_index(payload):
 if len(payload) == 40:
  index = int(payload[2:8],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 40:
  increment_hex = [payload[(8 + 2 * i):((8 + 2) + 2 * i)] for i in range(16)]
  list_increment = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i in
range(len(increment_hex)/2)]
  return list_increment
 else:
  return None
# Exemple :
payload = "21006f920178017b0181018c01980196019c019f"
index = decode_index(payload)
header = int(payload[0:2],16);
if header == 0x20 : # 10 minutes
 step = 10
elif header == 0x21 : # 15 minutes
 step = 15
elif header == 0x22 : # 1 hours
 step = 60
index = int(payload[2:8],16)
print "Index: " + str(index)
print "Step: " + str(step)
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print "Increments: " + str(list_increment)
 # Power conversion
 n = 60 / step
 list_power = []
 for i in range(len(list_increment)):
  list_power.append(round(n*list_increment[i],2)) # fills power list
 print("List of power values in W: "+str(list_power))
else:
 print("the payload has the wrong size")
```

6.6.2 *PHP Code example*

```
<?php
// return index from T1 payload
function decode_index($payload)
{
 $index = hexdec(substr($payload,2,6));
 return $index;
}
```

```
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 40)
 {
  for ($i = 0; $i < 16; $i++)
  {
   $increment_hex[$i] = substr($payload,8 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 40)
 {
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 8; $i++)
  {
   $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1]);
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Exemple :
$payload = "21006f920178017b0181018c01980196019c019f";
$index = decode_index($payload);
$header = hexdec(substr($payload,0,2));
if($header == 0x20) $step = 10;
if($header == 0x21) $step = 15;
if($header == 0x22) $step = 60;
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
 // Power conversion
 $n = 60 / $step;
 $list_power = [];
 $i = 0;
 foreach ($list_increment as $increment) //iteration over the increment
 {
  $list_power[$i] = round($increment*$n,2); //fills power list
  $i++;
 }
 echo "List of power values in W: \n";
 foreach ($list_power as $power)
 {
  echo $power. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

6.6.3 *JavaScript Code example*

```
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    if (payload.length == 40) {
        index = payload.substring(2, 8);
    }
    return parseInt(index, 16);
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var list_increment_hex = []
    if (payload.length == 40){
        for(i=0;i<16;i++){
            list_increment_hex.push(payload.substring((8+2*i),2+(8+2*i)))
        }
    }
    return list_increment_hex
}

//return power list from T1 payload
function decode_list_increment(payload) {
    var list_increment = []
    var list_increment_hex = []
    if (payload.length == 40){
        list_increment_hex = payload_to_hex(payload)
    }
    if(list_increment_hex.length == 16){
        for(i=0;i<8;i++){
            list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16))
        }
    }
    return list_increment
}

//return step from T1 payload
function decode_step(payload){
    var step = 0
    var header = null
    if(payload.length == 40){
        header = parseInt(payload.substring(0, 2), 16)
        if(header == 32) step = 10
        if(header == 33) step = 15
        if(header == 34) step = 60
    }
    return step
}
//Exemple
payload = "21006f920178017b0181018c01980196019c019f"
console.log(payload.length)
var index = null
var list_increment = []
var list_power = []
var step = 0

index = decode_index(payload)
step = decode_step(payload)
list_increment = decode_list_increment(payload)

if(index && list_increment.length == 8 && step > 0){
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    console.log("increment list: "+list_increment)
    //Power conversion
    n = 60/step
    for(i=0;i<list_increment.length;i++){
        list_power.push(list_increment[i]*n)
    }
    console.log("List of power values in W: "+list_power)
}
else{
    console.log("The payload has the wrong size")
}
```

7. Decoding FM432ir_nc_1mn messages

7.1 Types of messages

	Time-step
Message Type	**1 minute**
Data message (T1)	Every 15 minutes (15 x 1 minute)
Service message (T2)	Every 24 hours

FM432ir_nc_1mn generates two kinds of messages:

- T1: contains 15 or 20 successive values (depending on the type of meter) and the last index. It is generated every 15 or 20 minutes. The successive values are average power values in the case of an electromechanical meter. They are index increments in the case of an infrared meter (typically an mME meter).
- T2: contains additional technical information. It is generated once per day.

7.2 Data message (T1) structure in the case of an infrared meter (mME)

7.2.1 *Introduction*

The T1 data message transmits 1 **index** value (cumulated energy value) and 15 **increments**.

The **index value** is the cumulated energy since the start of the optical head.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent every 15 minutes.

7.2.2 *Timestamping*

Index value should be time stamped as follows: $t=t_{received}$

Each **increment** should be time stamped as follows: $t_i=t_{received} - ((15 - i)*Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,14}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements (here, 1 minute).

7.2.3 *Byte structure*

Byte	#1	#2	#3	#4	#5	#6	...	#11	#12	#13	#14	#15	#16	#17	#18	...	#39	#40	#41	#42
Signification	Header		Time-step	Sign	Index					Incr.(t0)		Incr.(t1)		Incr.(t2)		...	Incr(t13)		Incr(t14)	

7.2.4 *Header*

Header value (Hexa)	Signification
0xF02E	mME meter delivering E-SUM values (OBIS code 16.8.0 available)
0xF02F	mME meter delivering E-POS values (OBIS code 16.8.0 not available, but 1.8.0 available
0xF030	mME meter delivering E-NEG values (OBIS codes 16.8.0 and 1.8.0 not available, but 2.8.0 available)

Just after starting up, the sensor looks for specific OBIS codes in the following order:

- OBIS code 1.8.0 (E-POS): positive active energy (A+)
- OBIS code 16.8.0 (E-SUM): sum active energy without reverse blockade (A+ - A-)
- OBIS code 2.0.8 (E-NEG): negative active energy (A-)

7.2.5 Time-step

For FM432ir_nc_1mn product reference, Time-step is set to 1.

7.2.6 Sign

Indicates if the increment values are signed or unsigned.

Byte	**Value**	**Signification**
#4	00	Increment values are unsigned
	01	Increment values are signed

7.2.7 Index calculation

The index in Wh can be obtained this way:

Index = ((**Byte_4** * 2^56) + (**Byte_5** * 2^48) + (**Byte_6** * 2^40) + (**Byte_7** * 2^32) + (**Byte_8** * 2^24) + (**Byte_9** * 2^16) + (**Byte_10** * 2^8) + **Byte_11)** **/ 10**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

7.2.8 Increment values calculation

Increment values are labelled (t0) to (t19) and represent successive increments.

The formulas for obtaining increment values in Wh are:

Increment(t0) = ((Byte_12 * 2^8) + Byte_13) **/ 10**
Increment(t1) = ((Byte_14 * 2^8) + Byte_15) **/ 10**
Increment(t2) = ((Byte_16 * 2^8) + Byte_17) **/ 10**
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

Special values 0xFFFB, 0xFFFC, 0xFFFD, 0xFFFE and 0xFFFF are error codes to be communicated to Fludia for support.

7.2.9 Example

Given the following T1 message:

F02F 01 00 0000000000107900 0A0A 0A0B … 0E17 0C11

The decoded values should be:

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	...	#39	#40	#41	#42
Signification	**Header***		**Time step**	**Sign**	**Index**								**Incr.t0)**		**Incr.(t1)**		**...**	**Incr.(t13)**		**Incr.(t14)**	
Hexa value	F02F		01	00	00	00	00	00	00	10	79	00	0A	0A	0A	0B		0E	17	0C	11
Decoded value			01	00	107955.2 Wh								257.0 Wh		257.1 Wh			360.7 Wh		308.9 Wh	

7.3 Data message (T1) structure in the case of an electromechanical meter

7.3.1 Introduction

The T1 data message transmits 1 **index** value (cumulated energy value) and 20 **power** values.

A power value is the average power over 1 minute. The average power calculation is based on the elapsed time between two consecutive optical detections and involves interpolation in order to create 1 minute values. One optical detection occurs for each disk rotation, when the black mark (on the spinning disk) passes in front of the sensor. A multiplication factor (ratio or constant of the meter) is generally related to the meter and 1 detection can represent x Wh. In this case, the power values should be multiplied by this factor once received.

The T1 data message behaves as follows:

1. First message is sent 30 minutes after the sensor start-up.
2. Subsequent messages are sent periodically every 15 minutes.

Every T1 data message includes data related to a period of 15 minutes.

7.3.2 Timestamping

Index value should be time stamped as follows: $t=t_{received}$

Each **power** should be time stamped as follows: $t_i=t_{received} - delay - ((20 - i)*Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,19}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *delay is a delay of 10 minutes due to the power computation mechanism. It means that the message is sent 10 minutes after the time of the last 1 minute power. For example, if the message contains power values between 9:00am and 9:20am, the message will only be sent at 9:30am. This way, even if energy is low, there is time to wait for a late optical detection, necessary to compute average power, and attribute their share to the power values before sending the message.*
- *Time_step* is the time-step, i.e the interval between two measurements (here, 1 minute).

7.3.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	...	#40	#41	#42	#43	#44	#45
Signification	Header	Index				P(t0)		P(t1)		P(t2)		...	P(t17)		P(t18)		P(t19)	

7.3.4 Header

Header value (Hexa)	Signification
5b	1 min time step

7.3.5 Index calculation

The index can be obtained this way:

Index = (**Byte_2** * 2^24) + (**Byte_3** * 2^16) + (**Byte_4** * 2^8) + **Byte_5**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

7.3.6 Power values calculation

Power values are labelled (t0) to (t19).

The formulas for obtaining power values are:

Power(t0) = (Byte_6 * 2^8) + Byte_7
Power(t1) = (Byte_8 * 2^8) + Byte_9
Power(t2) = (Byte_10 * 2^8) + Byte_11
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

7.3.7 Example

Given the following T1 message:

50 0afdff00 068f 068f 0649 … 04b0 049b 04ac

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**...**	**#40**	**#41**	**#42**	**#43**	**#44**	**#45**
Signification	**Header***	**Index**				**P(t0)**		**P(t1)**		**P(t2)**		**...**	**P(t17)**		**P(t18)**		**P(t19)**	
Hexa value	50	0A	FD	FF	00	06	8F	06	8F	06	49		04	B0	04	9B	04	AC
Decoded value		184418048				1679		1679		1609			1200		1179		1196	

7.4 Service message (T2) structure in the case of an infrared meter (mME)

The T2 data message transmits technical information about the sensor (firmware version…) and more info about the measurements (scaler value, reminder of the current index…). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	…	#18
Signification	Header (F02A)		Time step	Type of measure	Forced Type?	firmware version	Sensor sensitivity	Scaler E-POS	Scaler E-SUM	Scaler E-NEG	Index		

Time-step

Byte	Value	Signification
#3	01	1-minute time step

Type of measure

Byte	Value	Signification
#4	00	E-POS values (OBIS code 1.8.0)
	01	E-SUM values (OBIS code 16.8.0)
	02	E-NEG values (OBIS code 2.8.0)

Forced Type?

Byte	Value	Signification
#5	00	Type of measure **hasn't been forced** via a downlink since starting up
	01	Type of measure **has been forced** via a downlink since starting up

Firmware version

Byte	Value	Signification
#6	0A	Optical head firmware version 1.0

Sensor sensitivity

Byte	Value	Signification
#7	00	highest sensitivity
	01	high intermediate sensitivity
	02	low intermediate sensitivity
	03	lowest sensitivity

Scaler E-POS

Byte	Value (signed hexa)	Signification
#8	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-POS (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3 ; 7F: OBIS code 1.8.0 not found)

Scaler E-SUM

Byte	Value (signed hexa)	Signification
#9	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-SUM (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3; 7F: OBIS code 16.8.0 not found)

Scaler E-NEG

Byte	Value (signed hexa)	Signification
#10	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-NEG (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3; 7F: OBIS code 2.8.0 not found)

Index

Bytes	Index calculation
#11…. #18	**Index** = ((**Byte_11** * 2^56) + (**Byte_12** * 2^48) + (**Byte_13** * 2^40) + (**Byte_14** * 2^32) + (**Byte_15** * 2^24) + (**Byte_16** * 2^16) + (**Byte_17** * 2^8) + **Byte_18)** **/ 10**

7.5 Service message (T2) structure in the case of an electromechanical meter

The T2 data message transmits technical information about the sensor (firmware, battery status indicator) and reminder of key values (index and time-step). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Signification	Header (4B)	Number of starts	Time synchronization	Not used	Optical head information	Index				Not used		Time step

Number of starts

Byte	**Signification**
#2	Number of starts since initial configuration (= number of times the sensor has been unplugged from the radio module)

Time synchronization information

Byte	**Bit**	**Signification**
#3	1 - 7	Jitter (in seconds)
	8	Synchro querying: 0 = no synchro querying 1 = synchro querying

Optical head information

Byte	**Bit**	**Signification**
#5	1 - 6	Firmware version
	7	Meter type: 0 = electromechanical meter
	8	Battery status: 0 = battery OK 1 = battery NOK (Battery voltage < 2.8 Volts)

Index

Bytes	**Index calculation**
#6, #7, #8, #9	**Index** = (**Byte_#6** * 2^24) + (**Byte_#7** * 2^16) + (**Byte_#8** * 2^8) + **Byte_#9**

Time-step

Byte	**Value (hexa)**	**Signification**
#12	02	1 minute time step

Fludia
Smart Energy Components

7.6 Code examples

We present examples of codes that perform the calculation for the index and for the power values, in the case of an infrared meter (mME). Examples of codes for the electromechanical meter are identical to the FM432e case (see chapter 5.4).

7.6.1 Python code example

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def twosComplement_hex(hexval, bits):
    val = int(hexval, 16)
    if val & (1 << (bits-1)):
        val -= 1 << bits
    return val
def decode_index(payload):
 if len(payload) == 84:
  signed = int(payload[6:8],16)
  if signed :
   index = twosComplement_hex(payload[8:24],64)
  else :
   index = int(payload[8:24],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 84:
  signed = int(payload[6:8],16)
  increment_hex = [payload[(24 + 2 * i):((24 + 2) + 2 * i)] for i in range(30)]
  if signed :
   list_increment_before_error_check = [twosComplement_hex(increment_hex[i * 2]+increment_hex[i * 2
+ 1], 16) for i in range(int(len(increment_hex)/2))]
   #We need to remove error codes
   list_increment = [ float(x)/10 if x != -1 and x != -2 and x != -3 and x != -4 and x != -5 else
None for x in list_increment_before_error_check]
  else :
   list_increment_before_error_check = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i
in range(int(len(increment_hex)/2))]
   #We need to remove error codes
   list_increment = [float(x)/10 if x != 65535 and x != 65534 and x != 65533 and x != 65532 and x !=
65531 else None for x in list_increment_before_error_check]
  return list_increment
 else:
  return None
# Exemple :
payload = "F02E0101FFFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB00080009000A000B000C000D000E000F"
index = decode_index(payload)
header = int(payload[2:4],16);
if header == 0x2E :
 obis = "E_SUM"
elif header == 0x2F :
 obis = "E_POS"
elif header == 0x30 :
 obis = "E_NEG"
step = int(payload[4:6],16)
print("Index: " + str(index))
print("Step: " + str(step))
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print("Increments: " + str(list_increment))
 # Power conversion
 n = 60 / step
 list_power = []
 for i in range(len(list_increment)):
  if list_increment[i] != None :
   list_power.append(round(n*list_increment[i],2)) # fills power list
  else :
   list_power.append(None)
 print("List of power values in W: "+str(list_power))
else:
 print("the payload has the wrong size")
```

7.6.2 PHP code example

```php
<?php
// return index from T1 payload
function decode_index($payload)
{
 $signed = hexdec(substr($payload,6,2));
 if($signed){
  //s is for signed values on 16 bits see https://www.php.net/manual/fr/function.pack.php
  $str = substr($payload,8,16);
  $temp = unpack('J', pack("H*", $str));
  $index = reset($temp);
 }else{
  $index = hexdec(substr($payload,8,16));
 }
 return $index;
}
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 84)
 {
  for ($i = 0; $i < 30; $i++)
  {
   $increment_hex[$i] = substr($payload,24 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 84)
 {
  $signed = hexdec(substr($payload,6,2));
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 15; $i++)
  {
   if($increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFE"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFD"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFC"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFB"){
     //ERROR CODE NO VALUE
     $list_increment[] = null;
   } else {
     if($signed){
     //s is for signed values on 16 bits see https://www.php.net/manual/fr/function.pack.php
     $temp = unpack("s", pack("s", hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1])));
     $incr = reset($temp);
     $list_increment[] = $incr/10;
     }else{
      $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1])/10;
     }
   }
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Exemple :
$payload = "F02E0101FFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB00080009000A000B000C000D000E000F";
$index = decode_index($payload);
$header = hexdec(substr($payload,2,2));
if($header == 0x2E) $obis = "E_SUM";
if($header == 0x2F) $obis = "E_POS";
if($header == 0x30) $obis = "E_NEG";
$step = hexdec(substr($payload,4,2));
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
```

```
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
 // Power conversion
 $n = 60 / $step;
 $list_power = [];
 $i = 0;
 foreach ($list_increment as $increment) //iteration over the increment
 {
  if($increment != null) $list_power[$i] = round($increment*$n,2); //fills power list
  else $list_power[$i] = null;
  $i++;
 }
 echo "List of power values in W: \n";
 foreach ($list_power as $power)
 {
  echo $power. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

7.6.3 JavaScript code example

```
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    var signed = parseInt(payload.substring(6, 8), 16)
    if(signed){
        index = BigInt.asIntN(64, "0x"+payload.substring(8,24))
    }
    else{
        index = BigInt.asUintN(64, "0x"+payload.substring(8,24))
    }
    return index;
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var power_list_hex = []
    for(i=0;i<30;i++){
        power_list_hex.push(payload.substring((24+2*i),2+(24+2*i)))
    }
    return power_list_hex
}

//return power list from T1 payload
function decode_increment(payload) {
    var list_increment = []
    var list_increment_hex = payload_to_hex(payload)
    var signed = parseInt(payload.substring(6, 8), 16)
    for(i=0;i<15;i++){
        if(list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFE" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFD" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFB" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFC"){
            list_increment.push(null)
        }
        else{
            if(signed){
                val = parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16)
                if ((val & 0x8000) > 0) {
                val = val - 0x10000;
                }
                list_increment.push(val/10)
            }
            else{
                list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1],
16)/10)
            }
        }
    }
    return list_increment
}
//Exemple
payload = "F02E0101FFFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB00080009000A000B000C000D000E000F"
console.log(payload.length)
if(payload.length == 84){
    var index = null
    var list_increment = []
    var list_power = []
    var step = 0

    var header = parseInt(payload.substring(2,4), 16)
    if(header == 46) obis = "E_SUM"
    if(header == 47) obis = "E_POS"
    if(header == 48) obis = "E_NEG"

    index = decode_index(payload)
    step = parseInt(payload.substring(4,6), 16)
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    list_increment = decode_increment(payload)
    console.log("increment list: ")
    console.log(list_increment)
    n = 60/step
    for(i=0;i<list_increment.length;i++){
        if(list_increment[i] != null){
            list_power.push(list_increment[i]*n)
        }
        else{
            list_power.push(null)
        }
    }
    console.log("List of power values in W: ")
    console.log(list_power)
}
else{
    console.log("The payload has the wrong size")
}
```

8. Decoding FM432ir_nc_15mn

8.1 Types of messages

	Time-step
Message Type	15 minutes
Data message (T1)	Every 2 hours (8 x 15 minutes)
Service message (T2)	Every 24 hours

FM432ir_nc_15mn generates two kinds of messages:

- T1: contains eight successive values and the last index. It is generated every 2 hours. The successive values are index increments.
- T2: contains additional technical information. It is generated once per day.

8.2 Data message (T1) structure in the case of an infrared meter (mME)

8.2.1 Introduction

The T1 data message transmits 1 **index value** (cumulated energy value) and 8 **increments**.

The **index value** is the cumulated energy since the start of the optical head.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent every 2 hours for 15-minutes time-step sensors.

8.2.2 Timestamping

Index value should be time stamped as follows: $t=t_{received}$

Each **increment** should be time stamped as follows: $t_i=t_{received} - ((8 - i)*Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,7}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements.

8.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	...	#11	#12	#13	#14	#15	#16	#17	#18	...	#25	#26	#27	#28
Signification	Header		Time-step	Sign	Index					Incr.(t0)		Incr.(t1)		Incr.(t2)		...	Incr.(t6)		Incr.(t7)	

8.2.4 Header

Header value (Hexa)	Signification
0xF02E	mME meter delivering E-SUM values (OBIS code 16.8.0 available)
0xF02F	mME meter delivering E-POS values (OBIS code 16.8.0 not available, but 1.8.0 available
0xF030	mME meter delivering E-NEG values (OBIS code 16.8.0 and 1.8.0 not available, but 2.8.0 available

Just after starting up, the sensor looks for specific OBIS codes in the following order:

- OBIS code 1.8.0 (E-POS): positive active energy (A+)
- OBIS code 16.8.0 (E-SUM): sum active energy without reverse blockade (A+ - A-)
- OBIS code 2.0.8 (E-NEG): negative active energy (A-)

8.2.5 Time-step

For FM432ir_nc_15mn product reference, Time-step is set to 0F.

8.2.6 Sign

Indicates if the increment values are signed or unsigned.

Byte	Value	Signification
#4	00	Increment values are unsigned
	01	Increment values are signed

8.2.7 Index calculation

The index in Wh can be obtained this way:

Index = ((**Byte_4** * 2^56) + (**Byte_5** * 2^48) + (**Byte_6** * 2^40) + (**Byte_7** * 2^32) + (**Byte_8** * 2^24) + (**Byte_9** * 2^16) + (**Byte_10** * 2^8) + **Byte_11)** **/ 10**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

8.2.8 Increment value calculation

Increment values are labelled (t0) to (t7) and represent successive increments.

The formulas for obtaining increment values in Wh are:

Increment(t0) = ((Byte_12 * 2^8) + Byte_13) **/ 10**
Increment(t1) = ((Byte_14 * 2^8) + Byte_15) **/ 10**
Increment(t2) = ((Byte_16 * 2^8) + Byte_17) **/ 10**
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

Special values 0xFFFB, 0xFFFC, 0xFFFD, 0xFFFE and 0xFFFF are error codes to be communicated to Fludia for support.

8.2.9 *Example*

Given the following T1 message for a 15 minutes time-step sensor:

F02F 0F 00 0000000000107900 0A0A 0A0B … 0E17 0C11

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**...**	**#25**	**#26**	**#27**	**#28**
Signification	**Header***		**Time step**	**Sign**	**Index**								**Incr.t0)**		**Incr.(t1)**		**...**	**Incr.(t13)**		**Incr.(t14)**	
Hexa value	F02F		0F	00	00	00	00	00	00	10	79	00	0A	0A	0A	0B		0E	17	0C	11
Decoded value			15	00	107955.2 Wh								257.0 Wh		257.1 Wh			360.7 Wh		308.9 Wh	

8.3 Data message (T1) structure in the case of an electromechanical meter

8.3.1 *Introduction*

The T1 data message transmits **1 index value** and **8 increments**.

The **index value** is the cumulated number of optical detections since the start of the sensor. One optical detection occurs for each disk rotation, when the black mark (on the spinning disk) passes in front of the sensor. **Important:** a multiplication factor (ratio or constant of the meter) is generally related to the meter and 1 detection can represent xWh.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent every 2 hours for 15-minutes time-step sensors.

8.3.2 *Timestamping*

Index value should be time stamped as follows: $t = t_{received}$

Each **increment** should be time stamped as follows: $t_i = t_{received} - ((8 - i) * Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,7}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements.

8.3.3 *Byte structure*

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20	#21
Signification	Header*	Index				Incr. (t0)		Incr. (t1)		Incr. (t2)		Incr. (t3)		Incr. (t4)		Incr. (t5)		Incr. (t6)		Incr. (t7)	

8.3.4 *Header*

*** Header values (Hexa)**	**Signification**
49	15-min time step

8.3.5 *Index calculation*

The index can be obtained this way:

Index = (**Byte_2** * 2^24) + (**Byte_3** * 2^16) + (**Byte_4** * 2^8) + **Byte_5**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

8.3.6 *Increment calculation*

Increment values are labelled (t0) to (t7) and represent successive increments.

The formulas for obtaining increment values are:

Increment(t0) = (Byte_6* 2^8) + Byte_7
Increment(t1) = (Byte_8 * 2^8) + Byte_9
Increment(t2) = (Byte_10 * 2^8) + Byte_11
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

8.3.7 *Example*

Given the following T1 message for a 15 minutes time-step sensor:

49 005A962B 0035 0B34 0B34 0A1F 00A1 007B 2206 1968

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**	**#21**
Signification	**Header***	**Index**				**Incr. (t0)**		**Incr. (t1)**		**Incr. (t2)**		**Incr. (t3)**		**Incr. (t4)**		**Incr. (t5)**		**Incr. (t6)**		**Incr. (t7)**	
Hexa value	49	00	5A	96	2B	00	35	0B	34	0B	34	0A	1F	00	A1	00	7B	22	06	19	68
Decoded value		5936683				53		2868		2868		2591		161		123		8710		6504	

8.4 Service message (T2) structure in the case of an infrared meter (mME)

The T2 data message transmits technical information about the sensor (firmware version…) and more info about the measurements (scaler value, reminder of the current index…). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	…	#18
Signification	Header (F02A)		Time step	Type of measure	Forced Type?	firmware version	Sensor sensitivity	Scaler E-POS	Scaler E-SUM	Scaler E-NEG	Index		

Time-step

Byte	**Value**	**Signification**
#3	0F	15 minutes time step

Type of measure

Byte	**Value**	**Signification**
#4	00	E-POS values (OBIS code 1.8.0)
	01	E-SUM values (OBIS code 16.8.0)
	02	E-NEG values (OBIS code 2.8.0)

Forced Type?

Byte	Value	Signification
#5	00	Type of measure **hasn't been forced** via a downlink since starting up
	01	Type of measure **has been forced** via a downlink since starting up

Firmware version

Byte	Value	Signification
#6	9	Optical head firmware version

Sensor sensitivity

Byte	Value	Signification
#7	00	highest sensitivity
	01	high intermediate sensitivity
	02	low intermediate sensitivity
	03	lowest sensitivity

Scaler E-POS

Byte	Value (signed hexa)	Signification
#8	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-POS (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3 ; 7F: OBIS code 1.8.0 not found)

Scaler E-SUM

Byte	Value (signed hexa)	Signification
#9	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-SUM (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3; 7F: OBIS code 16.8.0 not found)

Scaler E-NEG

Byte	Value (signed hexa)	Signification
#10	FF / 01 / 02 / 03 / 7F	Multiplication factor that has been applied to index and increment values in case the type is E-NEG (FF: 10^-1 ; 01: 10^1 ; 02: 10^2 ; 03: 10^3; 7F: OBIS code 2.8.0 not found)

Index

Bytes	Index calculation in Wh
#11…. #18	**Index** = ((**Byte_11** * 2^56) + (**Byte_12** * 2^48) + (**Byte_13** * 2^40) + (**Byte_14** * 2^32) + (**Byte_15** * 2^24) + (**Byte_16** * 2^16) + (**Byte_17** * 2^8) + **Byte_18)** **/ 10**

8.5 Service message (T2) structure in the case of an electromechanical meter

The T2 data message transmits technical information about the sensor (firmware, battery status indicator) and reminder of key values (index and time-step). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Signification	Header (4B)	Number of starts	Time synchronization	Not used	Optical head information	Long index				Not used		Time step (see below)

Number of starts

Byte	**Signification**
#2	Number of starts since initial configuration (= number of times the sensor has been unplugged from the radio module)

Time synchronization information

Byte	**Bit**	**Signification**
#3	1 - 7	Jitter (in seconds)
	8	Synchro querying: 0 = no synchro querying 1 = synchro querying

Optical head information

Byte	**Bit**	**Signification**
#5	1 - 6	Firmware version
	7	Meter type: 0 = electromechanical meter
	8	Battery status: 0 = battery OK 1 = battery NOK (Battery voltage < 2.8 Volts)

Index

Bytes	**Index calculation**
#6, #7, #8, #9	**Index** = (**Byte_#6** * 2^24) + (**Byte_#7** * 2^16) + (**Byte_#8** * 2^8) + **Byte_#9**

Time step

Byte	**Value**	**Signification**
#12	03	15 minutes time step

8.6 Code examples

We present examples of codes that perform the calculation for the index and for the power values, in the case of an infrared meter (mME). Examples of codes for the electromechanical meter are identical to the FM432e case (see chapter 5.4).

8.6.1 Python code example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def twosComplement_hex(hexval, bits):
    val = int(hexval, 16)
    if val & (1 << (bits-1)):
        val -= 1 << bits
    return val
def decode_index(payload):
 if len(payload) == 56:
  signed = int(payload[6:8],16)
  if signed :
   index = twosComplement_hex(payload[8:24],64)
  else :
   index = int(payload[8:24],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 56:
  signed = int(payload[6:8],16)
  increment_hex = [payload[(24 + 2 * i):((24 + 2) + 2 * i)] for i in range(16)]
  if signed :
   list_increment_before_error_check = [twosComplement_hex(increment_hex[i * 2]+increment_hex[i * 2
+ 1], 16) for i in range(int(len(increment_hex)/2))]
   #We need to remove error codes
   list_increment = [float(x)/10 if x != -1 and x != -2 and x != -3 and x != -4 and x != -5 else
None for x in list_increment_before_error_check]
  else :
   list_increment_before_error_check = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i
in range(int(len(increment_hex)/2))]
   #We need to remove error codes
   list_increment = [float(x)/10 if x != 65535 and x != 65534 and x != 65533 and x != 65532 and x !=
65531 else None for x in list_increment_before_error_check]
  return list_increment
 else:
  return None
# Exemple :
payload = "F02E0F00FFFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB0008"
index = decode_index(payload)
header = int(payload[2:4],16);
if header == 0x2E :
 obis = "E_SUM"
elif header == 0x2F :
 obis = "E_POS"
elif header == 0x30 :
 obis = "E_NEG"
step = int(payload[4:6],16)
print("Index: " + str(index))
print("Step: " + str(step))
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print("Increments: " + str(list_increment))
 # Power conversion
 n = 60 / step
 list_power = []
 for i in range(len(list_increment)):
  if list_increment[i] != None :
   list_power.append(round(n*list_increment[i],2)) # fills power list
  else :
   list_power.append(None)
 print("List of power values in W: "+str(list_power))
else:
 print("the payload has the wrong size")
```

8.6.2 PHP code example

```php
<?php
// return index from T1 payload
function decode_index($payload)
{
 $signed = hexdec(substr($payload,6,2));
 if($signed){
  //s is for signed values on 16 bits see https://www.php.net/manual/fr/function.pack.php
  $str = substr($payload,8,16);
  $temp = unpack('J', pack("H*", $str));
  $index = reset($temp);
 }else{
  $index = hexdec(substr($payload,8,16));
 }
 return $index;
}
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 56)
 {
  for ($i = 0; $i < 16; $i++)
  {
   $increment_hex[$i] = substr($payload,24 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 56)
 {
  $signed = hexdec(substr($payload,6,2));
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 8; $i++)
  {
   if($increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFE"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFD"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFC"
       || $increment_hex[$i * 2].$increment_hex[$i * 2 + 1] == "FFFB"){
    //ERROR CODE NO VALUE
    $list_increment[] = null;
   } else {
    if($signed){
    //s is for signed values on 16 bits see https://www.php.net/manual/fr/function.pack.php
    $temp = unpack("s", pack("s", hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1])));
    $incr = reset($temp);
    $list_increment[] = $incr/10;
    }else{
     $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1])/10;
    }
   }
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Exemple :
$payload = "F02E0F01FFFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB0008";
$index = decode_index($payload);
$header = hexdec(substr($payload,2,2));
if($header == 0x2E) $obis = "E_SUM";
if($header == 0x2F) $obis = "E_POS";
if($header == 0x30) $obis = "E_NEG";
$step = hexdec(substr($payload,4,2));
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
 // Power conversion
 $n = 60 / $step;
 $list_power = [];
 $i = 0;
 foreach ($list_increment as $increment) //iteration over the increment
 {
  if($increment != null) $list_power[$i] = round($increment*$n,2); //fills power list
  else $list_power[$i] = null;
  $i++;
 }
 echo "List of power values in W: \n";
 foreach ($list_power as $power)
 {
  echo $power. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

8.6.3 *JavaScript code example*

```
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    var signed = parseInt(payload.substring(6, 8), 16)
    if(signed){
        index = BigInt.asIntN(64, "0x"+payload.substring(8,24))
    }
    else{
        index = BigInt.asUintN(64, "0x"+payload.substring(8,24))
    }
    return index;
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var power_list_hex = []
    for(i=0;i<16;i++){
        power_list_hex.push(payload.substring((24+2*i),2+(24+2*i)))
    }
    return power_list_hex
}

//return power list from T1 payload
function decode_increment(payload) {
    var list_increment = []
    var list_increment_hex = payload_to_hex(payload)
    var signed = parseInt(payload.substring(6, 8), 16)
    for(i=0;i<8;i++){
        if(list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFE" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFD" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFB" ||
list_increment_hex[i*2]+list_increment_hex[i*2+1] == "FFFC"){
            list_increment.push(null)
        }
        else{
            if(signed){
                val = parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16)
                if ((val & 0x8000) > 0) {
                val = val - 0x10000;
                }
                list_increment.push(val/10)
            }
            else{
                list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1],
16)/10)
            }
        }
    }
    return list_increment
}
//Exemple
payload = "F02E0F01FFFFFFFFFFFF1F5400010002FFF6FFFD00050006FFFB0008"
console.log(payload.length)
if(payload.length == 56){
    var index = null
    var list_increment = []
    var list_power = []
    var step = 0

    var header = parseInt(payload.substring(2,4), 16)
    if(header == 46) obis = "E_SUM"
    if(header == 47) obis = "E_POS"
    if(header == 48) obis = "E_NEG"

    index = decode_index(payload)
    step = parseInt(payload.substring(4,6), 16)
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    list_increment = decode_increment(payload)
    console.log("increment list: ")
    console.log(list_increment)
    n = 60/step
    for(i=0;i<list_increment.length;i++){
        if(list_increment[i] != null){
            list_power.push(list_increment[i]*n)
        }
        else{
            list_power.push(null)
        }
    }
    console.log("List of power values in W: ")
    console.log(list_power)
}
else{
    console.log("The payload has the wrong size")
}
```

9. Decoding FM432g_nc_10mn et FM432g_nc_15mn

9.1 Types of messages

Sensor version	10 minutes	15 minutes	1 hour
Data message (T1)	Every 80 minutes (8 x 10 minutes)	Every 2 hours (8 x 15 minutes)	Every 8 hours (8 x 1 hour)
Service message (T2)	Every 24 hours	Every 24 hours	Every 24 hours
Technical message 1 (TT1)	Every 24 hours	Every 24 hours	Every 24 hours

FM432e generates three kinds of messages:

- T1: contains the consumption data and is generated several times per day with a frequency determined by the sensor time-step.
- T2: contains useful information such as the sensor type, the firmware version, a long index. It is generated once per day.
- TT1: contains technical information (for technical feedback purposes) and is generated once per day.

9.2 Data message (T1) structure

9.2.1 Introduction

The T1 data message transmits **1 index value** and **8 increments**.

The **index value** is the cumulated number of optical detections since the start of the sensor. One optical detection occurs when the digit changes in front of the optical sensor. Since the sensor is positioned in front of the second digit from the right, it counts 1 when 10 dm3 are consumed (unless the meter has a different unit than dm3 for the last digit, which is rare). Therefore, the sensor multiplies the number of detections by ten, so that the values are in dm3, which is usually more convenient.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent:

- every 80 minutes for 10-minutes time-step sensors
- every 2 hours for 15-minutes time-step sensors
- every 8 hours for 1-hour time-step sensors

9.2.2 Timestamping

Index value should be time stamped as follows: $t = t_{received}$

Each **increment** should be time stamped as follows: $t_i = t_{received} - ((8 - i) * Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,7}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements.

9.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20
Signification	Header*	Index			Incr. (t0)		Incr. (t1)		Incr. (t2)		Incr. (t3)		Incr. (t4)		Incr. (t5)		Incr. (t6)		Incr. (t7)	

9.2.4 Header

*** Header values (Hexa)**	**Signification**
1D	10-min time step
1E	15-min time step
1F	1-hour time step

9.2.5 Index calculation

The index can be obtained this way:

Index = (**Byte_2** * 2^16) + (**Byte_3** * 2^8) + **Byte_4**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

9.2.6 Increment calculation

Increment values are labelled (t0) to (t7) and represent successive increments.

The formulas for obtaining increment values are:

Increment(t0) = (Byte_5 * 2^8) + Byte_6
Increment(t1) = (Byte_7 * 2^8) + Byte_8
Increment(t2) = (Byte_9 * 2^8) + Byte_10
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

9.2.7 Example

Given the following T1 message: 1E006f920178017b0181018c01980196019c019f

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**
Signification	**Header***	**Index**			**Incr. (t0)**		**Incr. (t1)**		**Incr. (t2)**		**Incr. (t3)**		**Incr. (t4)**		**Incr. (t5)**		**Incr. (t6)**		**Incr. (t7)**	
Hexa value	1E	00	6F	92	01	78	01	7B	01	81	01	8C	01	98	01	96	01	9C	01	9F
Decoded value		28562			376		379		385		396		408		406		412		415	

9.3 Service message (T2) structure

The T2 data message transmits information about the sensor (optical head firmware, number of starts) and more info about the measurements (a longer index, and a reminder of the time-step). The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Signification	Header (10)	Number of starts	Time synchronization (see below)	Param. Id	Optical head information (see below)	Long index				Not used		Time step (see below)

Number of starts

Byte	Signification
#2	Number of starts since initial configuration (= number of times the sensor is unplugged from the radio module)

Time synchronization information

Byte	Bit	Signification
#3	1 - 7	Jitter (in seconds)
	8	Synchro querying (needs specific implementation on the server side): 0 = no synchro querying 1 = synchro querying

Optical head information

Byte	Bit	Signification
#5	1 - 6	Firmware version
	7	Meter type: 1 = gas meter
	8	not used

Long index

Bytes	Index calculation
#6, #7, #8, #9	**Index** = (**Byte_#6** * 2^24) + (**Byte_#7** * 2^16) + (**Byte_#8** * 2^8) + **Byte_#9**

Time step

Byte	Value	Signification
#12	00	10-minutes time-step
	03	15-minutes time-step
	01	1-hour time-step

9.4 Technical message (TT1) structure

Byte	Signification
#1	Header (hexa): 2E
#2-#22	Reserved technical information

9.5 Code examples

We present examples of codes that perform the calculation for the index and for the increments.

9.5.1 Python code example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def decode_index(payload):
 if len(payload) == 40:
  index = int(payload[2:8],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 40:
  increment_hex = [payload[(8 + 2 * i):((8 + 2) + 2 * i)] for i in range(16)]
  list_increment = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i in
range(len(increment_hex)/2)]
  return list_increment
 else:
  return None
# Example:
payload = "1E006f920178017b0181018c01980196019c019f"
index = decode_index(payload)
header = int(payload[0:2],16);
if header == 0x1D : # 10 minutes
 step = 10
elif header == 0x1E : # 15 minutes
 step = 15
elif header == 0x1F : # 1 hours
 step = 60
index = int(payload[2:8],16)
print "Index: " + str(index)
print "Step: " + str(step)
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print "Increments: " + str(list_increment)
else:
 print("the payload has the wrong size")
```

9.5.2 PHP Code example

```
<?php
// return index from T1 payload
function decode_index($payload)
{
 $index = hexdec(substr($payload,2,6));
 return $index;
}
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 40)
 {
  for ($i = 0; $i < 16; $i++)
  {
   $increment_hex[$i] = substr($payload,8 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 40)
 {
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 8; $i++)
  {
   $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1]);
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Example:
$payload = "1E006f920178017b0181018c01980196019c019f ";
$index = decode_index($payload);
$header = hexdec(substr($payload,0,2));
if($header == 0x1D) $step = 10;
if($header == 0x1E) $step = 15;
if($header == 0x1F) $step = 60;
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

9.5.3 *PHP Code example*

```
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    if (payload.length == 40) {
        index = payload.substring(2, 8);
    }
    return parseInt(index, 16);
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var list_increment_hex = []
    if (payload.length == 40){
        for(i=0;i<16;i++){
            list_increment_hex.push(payload.substring((8+2*i),2+(8+2*i)))
        }
    }
    return list_increment_hex
}

//return power list from T1 payload
function decode_list_increment(payload) {
    var list_increment = []
    var list_increment_hex = []
    if (payload.length == 40){
        list_increment_hex = payload_to_hex(payload)
    }
    if(list_increment_hex.length == 16){
        for(i=0;i<8;i++){
            list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16))
        }
    }
    return list_increment
}

//return step from T1 payload
function decode_step(payload){
    var step = 0
    var header = null
    if(payload.length == 40){
        header = parseInt(payload.substring(0, 2), 16)
        if(header == 29) step = 10
        if(header == 30) step = 15
        if(header == 31) step = 60
    }
    return step
}
//Exemple
payload = "1E006f920178017b0181018c01980196019c019f"
console.log(payload.length)
var index = null
var list_increment = []
var list_power = []
var step = 0

index = decode_index(payload)
step = decode_step(payload)
list_increment = decode_list_increment(payload)

if(index && list_increment.length == 8 && step > 0){
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    console.log("increment list: "+list_increment)
}
else{
    console.log("The payload has the wrong size")
}
```

10. Decoding FM432p-(a | n)_nc_1mn

10.1 Types of messages

	Time-step
Message Type	1 minute
Data message (T1)	Every 20 minutes (20 x 1 minute)
Service message (T2)	Every 24 hours

FM432p-(a | n)_nc_1mn generates two kinds of messages:

- T1: contains twenty successive values and the last index. It is generated every 20 minutes. The successive values are index increments.
- T2: contains additional technical information. It is generated once per day.

10.2 Data message (T1) structure

10.2.1 Introduction

The T1 data message transmits 1 **index** value (cumulated energy value) and **20 increment** values.

The **index** is the cumulated number of pulse detections since the start of the sensor.

An **increment** is the index difference between a time -step and the next time-step.

T1 data messages are sent every 20 minutes.

10.2.2 Timestamping

Index value should be time stamped as follows: $t=t_{received}$

Each **Increment** should be time stamped as follows: $t_i=t_{received} - (20 - i)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,19}.
- $t_{received}$ is the timestamp when T1 data message is received.

10.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	...	#39	#40	#41	#42	#43	#44
Signification	Header*	Index			Incr(t0)		Incr(t1)		Incr(t2)		...	Incr(t17)		Incr(t18)		Incr(t19)	

10.2.4 Header

*** Header value (Hexa)**	**Signification**
5C	FM432p with 1-min time step

10.2.5 Index calculation

The index can be obtained this way:

Index = (**Byte_2** * 2^16) + (**Byte_3** * 2^8) + **Byte_4**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

10.2.6 Increment values calculation

Increment values are labelled (t0) to (t19) and represent successive increments.

The formulas for obtaining increment values are:

Incr(t0) = (Byte_5 * 2^8) + Byte_6
Incr(t1) = (Byte_7 * 2^8) + Byte_8
Incr(t2) = (Byte_9 * 2^8) + Byte_10
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

10.2.7 Example

Given the following T1 message:

5c 0afdff 0000 0001 0000 ... 0001 0001 0000

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**...**	**#39**	**#40**	**#41**	**#42**	**#43**	**#44**
Signification	**Header***	**Index**			**P(t0)**		**P(t1)**		**P(t2)**		**...**	**P(t17)**		**P(t18)**		**P(t19)**	
Hexa value	5C	0A	FD	FF	00	00	00	01	00	00		00	01	00	01	00	00
Decoded value		720383			0		1		0			1		1		0	

10.3 Service message (T2) structure

The T2 data message transmits additional information: firmware version, long index, number of increment values in each T1 message, time-step value. The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Signification	Header (29)	Firmware version			Long index				number of increment values in each T1 message	Time-step

Firmware version

Bytes	**Value example (to be considered as ASCII)**	**Signification**
#2, #3, #4	33, 31, 36	33 is ASCII code for 3, 31 is ASCII code for 1, 36 is ASCII code for 6. So, in this example the firmware version is 3.1.6

Long index

Bytes	Index calculation
#5, #6, #7, #8	**Index** = (**Byte_#5** * 2^24) + (**Byte_#6** * 2^16) + (**Byte_#7** * 2^8) + **Byte_#8**

Number of increment values in T1

Byte	Value (Hexa)	Signification
#9	14	20 increment values in each T1 message

Time step

Byte	Value (Hexa)	Signification
#10	01	1-minute time step

10.4 Code examples

We present examples of codes that perform the calculation for the index and for the power values.

10.4.1 Python code example

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def decode_index(payload):
 if len(payload) == 88:
  index = int(payload[2:8],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 88:
  increment_hex = [payload[(8 + 2 * i):((8 + 2) + 2 * i)] for i in range(40)]
  list_increment = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i in
range(len(increment_hex)/2)]
  return list_increment
 else:
  return None
# Example:
payload = "
5c0afdff000000010000000000000000000000000003000300030002000100000001000100020001000100000"
index = decode_index(payload)
header = int(payload[0:2],16);
if header == 0x5C : # 1 minute
 step = 1
index = int(payload[2:8],16)
print "Index: " + str(index)
print "Step: " + str(step)
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print "Increments: " + str(list_increment)
else:
 print("the payload has the wrong size")
```

10.4.2 PHP Code example

```php
<?php
// return index from T1 payload
function decode_index($payload)
{
 $index = hexdec(substr($payload,2,6));
 return $index;
}
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 88)
 {
  for ($i = 0; $i < 40; $i++)
  {
   $increment_hex[$i] = substr($payload,8 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 88)
 {
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 20; $i++)
  {
   $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1]);
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Example:
$payload = "
5c0afdff00000001000000000000000000000000000300030003000200010000000100010002000100010000";
$index = decode_index($payload);
$header = hexdec(substr($payload,0,2));
if($header == 0x5C) $step = 1;
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

10.4.3 JavaScript Code example

```javascript
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    if (payload.length == 88) {
        index = payload.substring(2, 8);
    }
    return parseInt(index, 16);
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var list_increment_hex = []
    if (payload.length == 88){
        for(i=0;i<40;i++){
            list_increment_hex.push(payload.substring((8+2*i),2+(8+2*i)))
        }
    }
    return list_increment_hex
}
```

```
//return power list from T1 payload
function decode_list_increment(payload) {
    var list_increment = []
    var list_increment_hex = []
    if (payload.length == 88){
        list_increment_hex = payload_to_hex(payload)
    }
    if(list_increment_hex.length == 40){
        for(i=0;i<20;i++){
            list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16))
        }
    }
    return list_increment
}
//Exemple
payload = "
5c0afdff0000000100000000000000000000000000300030003000200010000000100010002000100010000"
console.log(payload.length)
var index = null
var list_increment = []
var step = 0
var header = parseInt(payload.substring(0,2), 16)

if(header == 92) step = 1

index = decode_index(payload)
list_increment = decode_list_increment(payload)

if(index && list_increment.length == 20 && step > 0){
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    console.log("increment list: "+list_increment)
}
else{
    console.log("The payload has the wrong size !")
}
```

11. Decoding FM432p-(a | n)_nc_10mn and FM432p-(a | n)_nc_15mn

11.1 Types of messages

Sensor version	10 minutes	15 minutes	1 hour
Data message (T1)	Every 80 minutes (8 x 10 minutes)	Every 2 hours (8 x 15 minutes)	Every 8 hours (8 x 1 hour)
Service message (T2)	Every 24 hours	Every 24 hours	Every 24 hours

FM432p generates two kinds of messages:

- T1: contains the consumption data and is generated several times per day with a frequency determined by the time-step related to the product reference.
- T2: contains additional information such as the firmware version and a long index. It is generated once per day.

11.2 Data message (T1) structure

11.2.1 Introduction

The T1 data message transmits **1 index value** and **8 increments**.

The **index value** is the cumulated number of input pulses since the product start-up.

An **increment** is the index difference between a time-step and the next time-step.

T1 data messages are sent:

- every 80 minutes for a 10-minutes time-step product
- every 2 hours for a 15-minutes time-step product
- every 8 hours for a 1-hour time-step product

11.2.2 Timestamping

Index value should be time stamped as follows: $t=t_{received}$

Each **increment** should be time stamped as follows: $t_i=t_{received} - ((8 - i)*Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,7}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two measurements.

11.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20
Signification	Header*	Index			Incr. (t0)		Incr. (t1)		Incr. (t2)		Incr. (t3)		Incr. (t4)		Incr. (t5)		Incr. (t6)		Incr. (t7)	

11.2.4 Header

* Header values (Hexa)	Signification
2B	10-min time step
2C	15-min time step
2D	1-hour time step

11.2.5 Index calculation

The index can be obtained this way:

Index = (**Byte_2** * 2^16) + (**Byte_3** * 2^8) + **Byte_4**

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

11.2.6 Increment calculation

Increment values are labelled (t0) to (t7) and represent successive increments.

The formulas for obtaining increment values are:

Increment(t0) = (Byte_5 * 2^8) + Byte_6
Increment(t1) = (Byte_7 * 2^8) + Byte_8
Increment(t2) = (Byte_9 * 2^8) + Byte_10
…

Make sure to first convert from hexadecimal to decimal: **Byte_i** <= hex2dec(**Byte_i**)

11.2.7 Example

Given the following T1 message:

2C006f920178017b0181018c01980196019c019f

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**
Signification	**Header***	**Index**			**Incr. (t0)**		**Incr. (t1)**		**Incr. (t2)**		**Incr. (t3)**		**Incr. (t4)**		**Incr. (t5)**		**Incr. (t6)**		**Incr. (t7)**	
Hexa value	2C	00	6F	92	01	78	01	7B	01	81	01	8C	01	98	01	96	01	9C	01	9F
Decoded value		28562			376		379		385		396		408		406		412		415	

11.3 Service message (T2) structure

The T2 data message transmits additional information: firmware version, long index, number of increment values in each T1 message, time-step value. The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Signification	Header (29)	Firmware version			Long index				number of increment values in each T1 message	Time-step

Firmware version

Bytes	Value example (to be considered as ASCII)	Signification
#2, #3, #4	33, 31, 36	33 is ASCII code for 3, 31 is ASCII code for 1, 36 is ASCII code for 6. So, in this example the firmware version is 3.1.6

Long index

Bytes	Index calculation
#5, #6, #7, #8	**Index** = (**Byte_#5** * 2^24) + (**Byte_#6** * 2^16) + (**Byte_#7** * 2^8) + **Byte_#8**

Number of increment values in T1

Byte	Value (Hexa)	Signification
#9	08	8 increment values in each T1 message

Time step

Byte	Value (Hexa)	Signification
#12	0A	10-minutes time step
	0F	15-minutes time step
	3C	1-hour time step (60 minutes)

11.4 Code examples

We present examples of codes that perform the calculation for the index and for the power values.

11.4.1 Python code example

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return index from T1 payload
def decode_index(payload):
 if len(payload) == 40:
  index = int(payload[2:8],16)
  return index
 else:
  return None
# return increments from T1 payload
def decode_increment(payload):
 if len(payload) == 40:
  increment_hex = [payload[(8 + 2 * i):((8 + 2) + 2 * i)] for i in range(16)]
  list_increment = [int(increment_hex[i * 2]+increment_hex[i * 2 + 1], 16) for i in
range(len(increment_hex)/2)]
  return list_increment
 else:
  return None
# Example:
payload = "2C006f920178017b0181018c01980196019c019f"
index = decode_index(payload)
header = int(payload[0:2],16);
if header == 0x2B : # 10 minutes
 step = 10
elif header == 0x2C : # 15 minutes
 step = 15
elif header == 0x2D : # 1 hours
 step = 60
index = int(payload[2:8],16)
print "Index: " + str(index)
print "Step: " + str(step)
list_increment = decode_increment(payload)
if list_increment is not None and index is not None:
 print "Increments: " + str(list_increment)
else:
 print("the payload has the wrong size")
```

11.4.2 PHP Code example

```php
<?php
// return index from T1 payload
function decode_index($payload)
{
 $index = hexdec(substr($payload,2,6));
 return $index;
}
// return increment_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 40)
 {
  for ($i = 0; $i < 16; $i++)
  {
   $increment_hex[$i] = substr($payload,8 + 2 * $i,2);
  }
 }
 return $increment_hex;
}
// return increments from T1 payload
function decode_increment($payload)
{
 if(strlen($payload) == 40)
 {
  $increment_hex = payload_to_hex($payload);
  $list_increment = [];
  for ($i = 0; $i < 8; $i++)
  {
   $list_increment[] = hexdec($increment_hex[$i * 2].$increment_hex[$i * 2 + 1]);
  }
  return $list_increment;
 }
 else
 {
  return null;
 }
}
// Example:
$payload = "2C006f920178017b0181018c01980196019c019f";
$index = decode_index($payload);
$header = hexdec(substr($payload,0,2));
if($header == 0x2B) $step = 10;
if($header == 0x2C) $step = 15;
if($header == 0x2D) $step = 60;
$list_increment = decode_increment($payload);
if ($list_increment != null && $index != null)
{
 echo "Index: ".$index."\n";
 echo "Step: ".$step."\n";
 foreach ($list_increment as $value)
 {
  echo $value. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

11.4.3 JavaScript Code example

```javascript
//return index from T1 payload
function decode_index(payload) {
    var index = null;
    if (payload.length == 40) {
        index = payload.substring(2, 8);
    }
    return parseInt(index, 16);
}

//return hexadecimal power list from T1 payload
function payload_to_hex(payload) {
    var list_increment_hex = []
    if (payload.length == 40){
        for(i=0;i<16;i++){
            list_increment_hex.push(payload.substring((8+2*i),2+(8+2*i)))
        }
    }
    return list_increment_hex
}

//return power list from T1 payload
function decode_list_increment(payload) {
    var list_increment = []
    var list_increment_hex = []
    if (payload.length == 40){
        list_increment_hex = payload_to_hex(payload)
    }
    if(list_increment_hex.length == 16){
        for(i=0;i<8;i++){
            list_increment.push(parseInt(list_increment_hex[i*2]+list_increment_hex[i*2+1], 16))
        }
    }
    return list_increment
}

//return step from T1 payload
function decode_step(payload){
    var step = 0
    var header = null
    if(payload.length == 40){
        header = parseInt(payload.substring(0, 2), 16)
        if(header == 43) step = 10
        if(header == 44) step = 15
        if(header == 45) step = 60
    }
    return step
}
//Exemple
payload = "2C006f920178017b0181018c01980196019c019f"
console.log(payload.length)
var index = null
var list_increment = []
var list_power = []
var step = 0

index = decode_index(payload)
step = decode_step(payload)
list_increment = decode_list_increment(payload)

if(index && list_increment.length == 8 && step > 0){
    console.log("Index: "+index)
    console.log("Step: "+step+" min")
    console.log("increment list: "+list_increment)
}
else{
    console.log("The payload has the wrong size !")
}
```

12. Decoding FM432t_nc_1mn

12.1 Types of messages

	Time-step
Message Type	1 minute
Data message (T1)	Every 20 minutes (20 x 1 minute)
Service message (T2)	Every 24 hours

FM432t_1mn generates two kinds of messages:

- T1: contains twenty successive values. It is generated every 20 minutes. The successive values are temperatures.
- T2: contains additional technical information. It is generated once per day.

All hexadecimal values are unsigned, except Temperature values (they are signed).

12.2 Data message (T1) structure

12.2.1 Introduction

The T1 data message transmits **20 Temperature** values.

T1 data messages are sent every 20 minutes.

12.2.2 Timestamping

Each **Temperature** should be time stamped as follows: $t_i = t_{received} - (20 - i)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,19}.
- $t_{received}$ is the timestamp when T1 data message is received.

12.2.3 Byte structure

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**...**	**#37**	**#38**	**#39**	**#40**	**#41**	**#42**
Signification	**Header**	**Time-step**	**Temp(t0)**		**Temp(t1)**		**Temp(t2)**		**...**	**Temp(t17)**		**Temp(t18)**		**Temp(t19)**	

12.2.4 Header

Header value (Hexa)	**Signification**
57	FM432t with 1-min time step

12.2.5 Time-step

Time-step (Hexa)	Signification
01	1 minute

12.2.6 Temperature values calculation

Temperature values are labelled (t0) to (t19) and represent successive increments.

The formulas for obtaining increment values are:

Temp(t0) = ((Byte_3 * 2^8) + Byte_4)/100
Temp(t1) = ((Byte_5 * 2^8) + Byte_6)/100
Temp(t2) = ((Byte_7 * 2^8) + Byte_8)/100
…

Make sure to first convert from hexadecimal (signed) to decimal: **Byte_i** <= hex2dec(**Byte_i**)

12.2.7 Example

Given the following T1 message:

57 01 06f5 0708 0714 … 0484 046b 044c

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**...**	**#37**	**#38**	**#39**	**#40**	**#41**	**#42**
Signification	**Header***	**Time-step**	**Temp(t0)**		**Temp(t1)**		**Temp(t2)**		**...**	**Temp(t17)**		**Temp(t18)**		**Temp(t19)**	
Hexa value	57	01	06	F5	07	08	07	14		04	84	04	6B	04	4C
Decoded value			17.81°C		18.00°C		18.12°C			11.56°C		11.31°C		11.00°C	

12.3 Service message (T2) structure

The T2 data message transmits additional information such as the minimum and maximum Temperature over the last 24 hours. The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9
Signification	Header (58)	Max Temp		Min Temp		Max Temp variation		number of values in each T1 message	Time-step

Max Temp

Bytes	Value example (signed hexadecimal)	Signification
#2, #3	07BE	Maximum Temperature over the last 24 hours. Convert to decimal and divide by 100. Example: 07BE => 19.82°C

Min Temp

Bytes	Value example (signed hexadecimal)	Signification
#4, #5	FF06	Minimum Temperature over the last 24 hours. Convert to decimal and divide by 100. Example: FF06 => -2.50°C

Fludia
Smart Energy Components

Max Temp variation

Bytes	Value example (signed hexadecimal)	Signification
#6, #7	FDEA	Maximum Temperature variation between two successive measurements over the last 24 hours. Convert to decimal and divide by 100. Example: FF06 => -5.34°C (temperature drop of 5.34°C in one minute)

Number of values in each T1 message

Byte	Value (Hexa)	Signification
#8	14	20 values in each T1 message

Time step

Byte	Value (Hexa)	Signification
#9	01	1-minute time step

12.4 Code examples

We present examples of codes that perform the calculation for the temperature.

12.4.1 Python code example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return temperatures from T1 payload
def decode_temp(payload):
 if len(payload) == 84:
  temp_hex = [payload[(4 + 2 * i):((4 + 2) + 2 * i)] for i in range(40)]
  list_temp = [float(int(temp_hex[i * 2]+temp_hex[i * 2 + 1], 16))/100 for i in
range(len(temp_hex)/2)]
  return list_temp
 else:
  return None
# Example:
payload = "570106f507080714071a072d070806b60665061a05dc059d056b0533050704e204c204a30484046b044c"
header = int(payload[0:2],16);
step = int(payload[2:4],16)
print "Step: " + str(step)
list_temp = decode_temp(payload)
if list_temp is not None:
 print "Temperatures: " + str(list_temp)
else:
 print("the payload has the wrong size")
```

12.4.2 PHP Code example

```
<?php
// return temp_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 84)
 {
  for ($i = 0; $i < 40; $i++)
  {
   $temp_hex[$i] = substr($payload,4 + 2 * $i,2);
  }
 }
 return $temp_hex;
}
// return temperatures from T1 payload
function decode_temp($payload)
```

```
{
 if(strlen($payload) == 84)
 {
  $temp_hex = payload_to_hex($payload);
  $list_temp = [];
  for ($i = 0; $i < 20; $i++)
  {
   $list_temp[] = hexdec($temp_hex[$i * 2].$temp_hex[$i * 2 + 1])/100;
  }
  return $list_temp;
 }
 else
 {
  return null;
 }
}
// Example:
$payload = "570106f507080714071a072d070806b60665061a05dc059d056b0533050704e204c204a30484046b044c";
$header = hexdec(substr($payload,0,2));
$step = hexdec(substr($payload,2,2));
$list_temp = decode_temp($payload);
if ($list_temp != null)
{
 echo "Step: ".$step."\n";
 foreach ($list_temp as $value)
 {
  echo $value. "\n";
 }
}
else
{
 echo "the payload has the wrong size \n";
}
?>
```

12.4.3 JavaScript Code example

```
//return hexadecimal temperature list from T1 payload
function payload_to_hex(payload) {
    var temp_list_hex = []
    if (payload.length == 84){
        for(i=0;i<40;i++){
            temp_list_hex.push(payload.substring((4+2*i),2+(4+2*i)))
        }
    }
    return temp_list_hex
}
//return temperature list from T1 payload
function decode_temp_list(payload) {
    var temp_list = []
    var temp_list_hex = []
    if (payload.length == 84){
        temp_list_hex = payload_to_hex(payload)
    }
    if(temp_list_hex.length == 40){
        for(i=0;i<20;i++){
            temp_list.push(parseInt(temp_list_hex[i*2]+temp_list_hex[i*2+1], 16)/100)
        }
    }
    return temp_list
}
//Exemple
payload = "570106f507080714071a072d070806b60665061a05dc059d056b0533050704e204c204a30484046b044c"
var list_temp = []
step = parseInt(payload.substring(2,4),16)
if(payload.length==84){
    list_temp = decode_temp_list(payload)
    console.log("Temperatures: "+list_temp)

}
else{
    console.log("The payload has the wrong size")
}
```

13. Decoding FM432t_nc_10mn and FM432t_nc_15mn

13.1 Types of messages

Sensor version	10 minutes	15 minutes
Data message (T1)	Every 80 minutes (8 x 10 minutes)	Every 2 hours (8 x 15 minutes)
Service message (T2)	Every 24 hours	Every 24 hours

FM432t generates two kinds of messages:

- T1: contains the Temperature data and is generated several times per day with a frequency determined by the time-step related to the product reference.
- T2: contains additional technical information. It is generated once per day.

13.2 Data message (T1) structure

13.2.1 Introduction

The T1 data message transmits **8 Temperature** values.

Each of these Temperature values is an average of several measurements. Individual measurements are performed every minute. If the data time-step is 15 minutes (for example), 15 successive individual measurements are averaged to create one 15 minute Temperature value.

T1 data messages are sent:

- every 80 minutes for a 10-minutes time-step product
- every 2 hours for a 15-minutes time-step product.

13.2.2 Timestamping

Each **Temperature** should be time stamped as follows: $t_i = t_{received} - ((8 - i) * Time\_step)$

Where:

- t_i is the timestamp for one element of the data message i ϵ {0,7}.
- $t_{received}$ is the timestamp when T1 data message is received.
- *Time_step* is the time-step, i.e the interval between two values.

13.2.3 Byte structure

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18
Signification	Header	Time-step	Temp (t0)		Temp (t1)		Temp (t2)		Temp (t3)		Temp (t4)		Temp (t5)		Temp (t6)		Temp (t7)	

13.2.4 Header

Header value (Hexa)	**Signification**
57	FM432t

13.2.5 *Time-step*

Time-step (Hexa)	Signification
0A	10 minutes
0F	15 minutes

13.2.6 *Temperature calculation*

Temperature values are labelled (t0) to (t7).

The formulas for obtaining Temperature values are:

Increment(t0) = ((Byte_3 * 2^8) + Byte_4)/100
Increment(t1) = ((Byte_5 * 2^8) + Byte_6)/100
Increment(t2) = ((Byte_7 * 2^8) + Byte_8)/100
…

Make sure to first convert from hexadecimal (signed) to decimal: **Byte_i** <= hex2dec(**Byte_i**)

13.2.7 *Example*

Given the following T1 message:

57 01 06f5 0708 0714 … 0484 046b 044c

The decoded values should be:

Byte	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**...**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**
Signification	**Header***	**Time-step**	**Temp(t0)**		**Temp(t1)**		**Temp(t2)**		**...**	**Temp(t5)**		**Temp(t6)**		**Temp(t7)**	
Hexa value	57	01	06	F5	07	08	07	14		04	84	04	6B	04	4C
Decoded value			17.81°C		18.00°C		18.12°C			11.56°C		11.31°C		11.00°C	

13.3 Service message (T2) structure

The T2 data message transmits additional information such as the minimum and maximum Temperature over the last 24 hours. The T2 data message is sent on sensor start and then every 24 hours.

Byte	#1	#2	#3	#4	#5	#6	#7	#8	#9
Signification	Header (58)	Max Temp		Min Temp		Max Temp variation		number of values in each T1 message	Time-step

Max Temp

Bytes	Value example (Hexa)	Signification
#2, #3	07BE	Maximum Temperature over the last 24 hours. Convert to decimal and divide by 100. Example: 07BE => 19.82°C

Min Temp

Bytes	Value example (Hexa)	Signification
#4, #5	FF06	Minimum Temperature over the last 24 hours. Convert to decimal and divide by 100. Example: FF06 => -2.50°C

Max Temp variation

Bytes	Value example (Hexa)	Signification
#6, #7	FDEA	Maximum Temperature variation between two successive measurements over the last 24 hours. Convert to decimal and divide by 100. Example: FF06 => -5.34°C (temperature drop of 5.34°C in one minute)

Number of increment values in T1

Byte	Value (Hexa)	Signification
#8	08	8 values in each T1 message

Time step

Byte	Value (Hexa)	Signification
#9	0A	10-minute time step
	0F	15-minute time step

13.4 Code examples

We present examples of codes that perform the calculation for the temperature.

13.4.1 *Python code example*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# return temperatures from T1 payload
def decode_temp(payload):
 if len(payload) == 36:
  temp_hex = [payload[(4 + 2 * i):((4 + 2) + 2 * i)] for i in range(16)]
  list_temp = [float(int(temp_hex[i * 2]+temp_hex[i * 2 + 1], 16))/100 for i in
range(len(temp_hex)/2)]
  return list_temp
 else:
  return None
# Example:
payload = "570a098809420947096009600979097909b5"
header = int(payload[0:2],16);
step = int(payload[2:4],16)
print "Step: " + str(step)
list_temp = decode_temp(payload)
if list_temp is not None:
 print "Temperatures: " + str(list_temp)
else:
 print("the payload has the wrong size")
```

13.4.2 *PHP Code example*

```
<?php
// return temp_hex from T1 payload
function payload_to_hex($payload)
{
 if(strlen($payload) == 36)
 {
  for ($i = 0; $i < 16; $i++)
  {
   $temp_hex[$i] = substr($payload,4 + 2 * $i,2);
  }
 }
 return $temp_hex;
}
// return temperatures from T1 payload
function decode_temp($payload)
{
 if(strlen($payload) == 36)
 {
  $temp_hex = payload_to_hex($payload);
  $list_temp = [];
  for ($i = 0; $i < 8; $i++)
  {
   $list_temp[] = hexdec($temp_hex[$i * 2].$temp_hex[$i * 2 + 1])/100;
  }
  return $list_temp;
 }
 else
 {
  return null;
 }
}
// Example:
$payload = "570a098809420947096009600979097909b5";
$header = hexdec(substr($payload,0,2));
$step = hexdec(substr($payload,2,2));
$list_temp = decode_temp($payload);
if ($list_temp != null)
{
 echo "Step: ".$step."\n";
 foreach ($list_temp as $value)
 {
  echo $value. "\n";
 }
}
```

```
else
{
 echo "the payload has the wrong size \n";
}
?>
```

13.4.3 JavaScript Code example

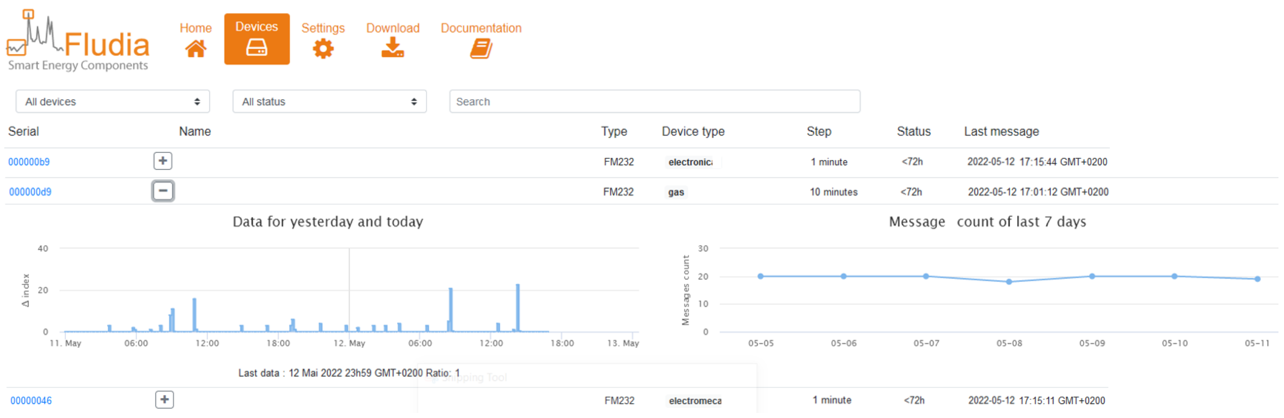
```
//return hexadecimal temperature list from T1 payload
function payload_to_hex(payload) {
    var temp_list_hex = []
    if (payload.length == 36){
        for(i=0;i<16;i++){
            temp_list_hex.push(payload.substring((4+2*i),2+(4+2*i)))
        }
    }
    return temp_list_hex
}
//return temperature list from T1 payload
function decode_temp_list(payload) {
    var temp_list = []
    var temp_list_hex = []
    if (payload.length == 36){
        temp_list_hex = payload_to_hex(payload)
    }
    if(temp_list_hex.length == 16){
        for(i=0;i<8;i++){
            temp_list.push(parseInt(temp_list_hex[i*2]+temp_list_hex[i*2+1], 16)/100)
        }
    }
    return temp_list
}
//Exemple
payload = "570a098809420947096009600979097909b5"
var list_temp = []
step = parseInt(payload.substring(2,4),16)
if(payload.length==36){
    list_temp = decode_temp_list(payload)
    console.log("Temperatures: "+list_temp)

}
else{
    console.log("The payload has the wrong size")
}
```

14. Accessing the dashboard to check communication and data

In case a callback has been set up to forward messages to Fludia data server, it is possible to access the dashboard with a Web browser, by typing the following URL: https://fm400-api.fludia.com/console

The login and password have been provided by email at the moment of purchase.

The dashboard displays the list of sensors and some indicators (Version number, time-step, elapsed time since last connection…).



Serial	Name	Type	Device type	Step	Status	Last message
000000b9	+	FM232	electronic	1 minute	<72h	2022-05-12 17:15:44 GMT+0200
000000d9	−	FM232	gas	10 minutes	<72h	2022-05-12 17:01:12 GMT+0200
00000046	+	FM232	electromeca	1 minute	<72h	2022-05-12 17:15:11 GMT+0200

Last data : 12 Mai 2022 23h59 GMT+0200 Ratio: 1

By clicking on the + sign, you can unstack two graphs, one showing measurement data and the other message counts.

By clicking on the ID (Serial) number, you can access a more detailed page, where you can graphically navigate the measurement data, download corresponding data files and have a look at the raw messages.

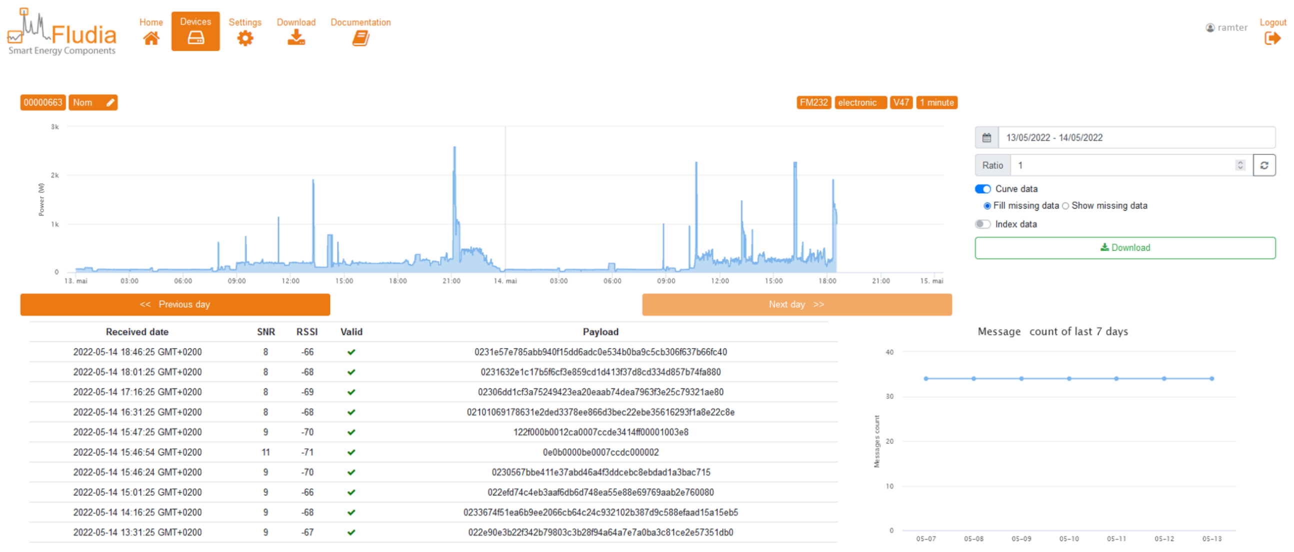
The top part of this page shows a graph with the measurement data. By default, the displayed period is two days, but a different period can be selected in the calendar.

It is possible to go back and forward in time, one day at a time, by clicking on the “Previous day” or “Next day” button.

The type of data on display can be chosen and be either curve data (could be called interval data, can refer to electric power or gas volume for example) or index data (cumulative quantity, can refer to cumulated energy or cumulated volume for example).

There are two options for curve data: “fill missing data” or “show missing data”. The “fill missing data” option creates values by spreading the energy evenly over the missing points. The “show missing value” does not create new values and the mssing periods appear clearly as blanks on the graph.

Index data is always provided without filling missing values.



Received date	SNR	RSSI	Valid	Payload
2022-05-14 18:46:25 GMT+0200	8	-66	✓	0231e57e785abb940f15dd6adc0e534b0ba9c5cb306f637b66fc40
2022-05-14 18:01:25 GMT+0200	8	-68	✓	0231632e1c17b5f6cf3e859cd1d413f37d8cd334d857b74fa880
2022-05-14 17:16:25 GMT+0200	8	-69	✓	02306dd1cf3a75249423ea20eaab74dea7963f3e25c79321ae80
2022-05-14 16:31:25 GMT+0200	8	-68	✓	02101069178631e2ded3378ee866d3bec22ebe35616293f1a8e22c8e
2022-05-14 15:47:25 GMT+0200	9	-70	✓	122f000b0012ca0007ccde3414ff00001003e8
2022-05-14 15:46:54 GMT+0200	11	-71	✓	0e0b0000be0007ccdc000002
2022-05-14 15:46:24 GMT+0200	9	-70	✓	0230567bbe411e37abd46a4f3ddcebc8ebdad1a3bac715
2022-05-14 15:01:25 GMT+0200	9	-66	✓	022efd74c4eb3aaf6db6d748ea55e88e69769aab2e760080
2022-05-14 14:16:25 GMT+0200	9	-68	✓	0233674f51ea6b9ee2066cb64c24c932102b387d9c588efaad15a15eb5
2022-05-14 13:31:25 GMT+0200	9	-67	✓	022e90e3b22f342b79803c3b28f94a64a7e7a0ba3c81ce2e57351db0

By clicking on the “download” button a csv file is being created and downloaded containing the measurement data on display, with the selected option. Once opened in a spreadsheet, measurement data looks like the example below.

Timestamp (s)	Power without ratio (W)	Power with ratio (W)	Ratio	Type	Step (minute)	Date
1652468580	272	272	1	FM232	1	2022-05-13 21:03:00 GMT+0200
1652468640	647	647	1	FM232	1	2022-05-13 21:04:00 GMT+0200
1652468700	2079	2079	1	FM232	1	2022-05-13 21:05:00 GMT+0200
1652468760	1749	1749	1	FM232	1	2022-05-13 21:06:00 GMT+0200
1652468820	1409	1409	1	FM232	1	2022-05-13 21:07:00 GMT+0200
1652468880	2580	2580	1	FM232	1	2022-05-13 21:08:00 GMT+0200

15. Retrieving data from the server with the API

In case a callback has been set up to forward messages to Fludia data server, it is possible to use Fludia API.

The access is done through HTTPS with the following URL:

https://fm430-api.fludia.com/v1/API/

Each query to the API needs authentication using the Basic Auth access. The login and password are provided separately by email (they are the same as the ones used to access the dashboard).

Exploring the API before implementation.

If needed as a first step, there are several ways to explore the API:

a. CURL command line: curl -u "login:password" "https://fm430-api.fludia.com/v1/API/..."

b. Postman tool

c. swagger.fludia.com (list of requests and test capability)

15.1 Device list request

GET https://fm430-api.fludia.com/v1/API/FM400

returns the list of devices (Last 8 characters of the device ID), the timestamp of the last received message for each device and some additional attributes.

Result example:

```
[
  {
    "Nom": "",
    "SerialNumber": "a000596c",
    "Type": "FM430",
    "Type_decodage": "V2",
    "Type_capteur": "Electronique",
    "Version_firmware": "21",
    "DerniereReceptionOK_TS": "1649066589",
    "Pas": "10 minutes",
    "Actif": "actif",
    "Pile_faible": "0",
    "Ratio": "1"
  },
  {
    "Nom": "",
    "SerialNumber": "a000596d",
    "Type": "FM430",
    "Type_decodage": "V2",
    "Type_capteur": "Electromécanique",
    "Version_firmware": "21",
    "DerniereReceptionOK_TS": "1649067662",
    "Pas": "10 minutes",
    "Actif": "actif",
    "Pile_faible": "0",
    "Ratio": "1"
  }
]
```

Attributes :

SerialNumber: last 8 characters of the device ID (device ID is found on the device front panel label, not to be confused with SN number… a bit confusing, but it comes from older ways to deal with device identification).

Type: FM430 refers to all LoRa devices (FM232x and FM432x).

Type_decodage: version of message decoding performed by the server (no use for normal API usage).

Type_capteur: Electronique (electronic electricity meter), Electromécanique (electromechanical electricity meter), IR (infrared SML german electricity meter), gaz (gas meter), pulse (pulse detection), Température (temperature).

Version_firmware: firmware version

DerniereReceptionOK_TS: timestamp of the last received message

Pas: measurement time step (1 minute, 10 minutes, 15 minutes, 1 heure)

Actif: “actif” signifies that the device has sent at least 1 message in the last three days (otherwise, it is “inactive”.

Pile faible: low battery indicator. Equals 1 when the battery voltage is under a threshold, 0 otherwise

(indicator available only for some of the sensors… if not available, the value is NULL)

Ratio: indicates the value of the constant that might be used to multiply the data (taken into account only in a specific request not presented in this document). This constant can be set through the API (for example to take into account the constant of an electromechanical meter in Wh per disk turn). Default value is 1.

15.2 Interval data request

GET
https://fm430-api.fludia.com/v1/API/pxm/**SerialNumber**[?limit=**n**[&tsDeb=**x**&tsFin=**x**]&show_missing=true]

Input data

SerialNumber	
limit	Maximum number of messages to be collected
&tsDeb=	First Timestamp
&tsFin=	Last Timestamp
show_missing	Option to process missing data

Returns the list of interval data, with related UTC timestamps.

&tsDeb= et &tsFin= are timestamps (in seconds) to filter data from tsDeb to tsFin (not mandatory).

&limit=n to limit to n last values.

show_missing chooses the way to process data. It is "false" by default.

Query « pxm »	**show_missing=false (default)**	**show_missing=true**
Missing radio message	Missing data are filled-up by averaged values	Missing data are not filled-up

Result example (first value is the TimeStamp, second value is the measurement value, without taking any ratio into account):

```
[
  [
    1628860800,
    57.2
  ],
  [
    1628861400,
    52.8
  ]
]
```

Depending on the sensor type (Type_capteur), the measurement values have different meaning:

- Electronique (optical reading of electronic electricity meter): the value is an average power (in Watt, to be multiplied by a constant if the meter comes with such a constant)

- Electromécanique (optical reading of electromechanical electricity meter): the value is an average power (in Watt, to be multiplied by a constant if the meter comes with such a constant)
- IR (optical reading of infrared SML german electricity meter): the value is an average power (in Watt)
- gaz (optical reading of gas meter): the value is a volume (in tens of dm3, if the sensor has been correctly positioned on the digit left to the dm3 digit)
- pulse (pulse detection): the value is the number of pulses detected over the time step
- Température (temperature): the value is the average temperature over the time step

15.3 Index data

GET https://fm430-api.fludia.com/v1/API/index_brut/**SerialNumber**[?limit=**n**[&tsDeb=**x**&tsFin=**x**]]

Input data

SerialNumber	
limit	Maximum number of messages to be collected
tsDeb	First Timestamp
tsFin	Last Timestamp

&tsDeb= et &tsFin= for timestamps (in seconds) to filter data from tsDeb to tsFin (not mandatory).

&limit=n to limit to n last values.

Result example (first value is the TimeStamp, second value is the index value):

```
  [
    1628860800,
    578956
  ],
  [
    1628861400,
    578958
  ]
]
```

Depending on the sensor type (Type_capteur), the index values have different meaning:

- Electronique (optical reading of electronic electricity meter): the value is the cumulative energy since installation (in Watt.hour, to be multiplied by a constant if the meter comes with such a constant)
- Electromécanique (optical reading of electromechanical electricity meter): the value is the cumulative energy since installation (in Watt.hour, to be multiplied by a constant if the meter comes with such a constant)
- IR (optical reading of infrared SML german electricity meter): the value is the cumulative energy provided by the meter (in Watt.hour)
- gaz (optical reading of gas meter): the value is the cumulative volume since installation (in tens of

dm3, if the sensor has been correctly positioned on the digit left to the dm3 digit)

- pulse (pulse detection): the value is the cumulative number of pulses since installation
- Température (temperature): the value is the average temperature over the time step

15.4 Message list

GET https://fm430-api.fludia.com/v1/API/trames?SerialNumber=**SerialNumber**[&limit=**n**&offset=**m**]

Input data

SerialNumber	
limit	Maximum number of messages to be collected
offset	Number of last messages to avoid

&limit=n to limit to n last messages. &offset=m to avoid the last m messages.

returns the last n messages received from the device:

```
[
  {
    "SerialNumber": "000017c5",
    "TsReception": 1649080309,
    "Valide": true,
    "Data":
"5b000615330fe30b120b030b660af7107e142a1600163015e40b870b1f0ec90be2067509df0daa0fca131016
1e",
    "Variables": {},
    "Rssi": -77,
    "Snr": 11
  },
  {
    "SerialNumber": "000017c5",
    "TsReception": 1649079109,
    "Valide": true,
    "Data":
"5b00061037095b04bf06d504bf04a307760a880c6a0c5a0c74074406800f870f7a151d18171a461be6191c14
14",
    "Variables": {},
    "Rssi": -77,
    "Snr": 8
  }
]
```

16. Contact

For further information or advice please contact us:

Fludia

support@fludia.com

01 83 64 13 94

4 ter rue Honoré d’Estienne d’Orves

92150 Suresnes, France

17. Annex A: product references and what they mean

FM432e: electricity meter optical reading

- Ref: FM432e_nc_1mn => no compression, average power measured over 1 minute; message sent every 20 minutes including 20 values
- Ref: FM432e_nc_10mn => no compression, average power measured over 10 minutes; message sent every 80 minutes including 8 values
- Ref: FM432e_nc_15mn => no compression, average power measured over 15 minutes; message sent every 120 minutes including 8 values

FM432ir: electricity infrared meter optical reading (SML protocol)

- Ref: FM432ir_nc_1mn => no compression, average power measured over 1 minute, message sent every 15 minutes including 15 values.
- Ref: FM432ir_nc_15mn => no compression, average power measured over 15 minutes, message sent every 120 minutes including 8 values

FM432g: gas meter optical reading (ATEX)

- Ref: FM432g_nc_10mn => no compression, volume measured over 10 minutes, message sent every 80 minutes including 8 values.
- Ref: FM432g_nc_15mn => no compression, volume measured over 15 minutes, message sent every 120 minutes including 8 values.

FM432p-a: pulse reading (ATEX)

- Ref: FM432p-a_nc_1mn => no compression, volume measured over 1 minute, message sent every 20 minutes including 20 values.
- Ref: FM432p-a_nc_10mn => no compression, volume measured over 10 minutes, message sent every 80 minutes including 8 values.
- Ref: FM432p-a_nc_15mn => no compression, volume measured over 15 minutes, message sent every 120 minutes including 8 values.

FM432p-n: pulse reading (non-ATEX)

- Ref: FM432p-n_nc_1mn => no compression, volume measured over 1 minute, message sent every 20 minutes including 20 values.
- Ref: FM432p-n_nc_10mn => no compression, volume measured over 10 minutes, message sent every 80 minutes including 8 values.
- Ref: FM432p-n_nc_15mn => no compression, volume measured over 15 minutes, message sent every 120 minutes including 8 values.

FM432t: temperature measurement

- Ref: FM432t_1mn => average temperature measured over 1 minute; message sent every 20 minutes including 20 values
- Ref: FM432t_10mn => average temperature measured over 10 minutes; message sent every 80 minutes including 8 values
- Ref: FM432t_15mn => average temperature measured over 15 minutes; message sent every 120 minutes including 8 values