TECHNICAL MANUAL

CAIRSENS PM

MICROSENSOR

- JULY 2020 -

111, Bd Robespierre / CS 80004 - 78304 Poissy Cedex 4 – France
Tél : +33(0)1 39 22 38 00 – Fax : +33(0)1 39 65 38 08 – www.envea.global

Index of pages

Pages	Dates	Pages	Dates	Pages	Dates
1	2020.07	21	2020.07		
2	2020.07	22	2020.07		
3	2020.07	23	2020.07		
4	2020.07	24	2020.07		
5	2020.07	25	2020.07		
6	2020.07	26	2020.07		
7	2020.07	27	2020.07		
8	2020.07	28	2020.07		
9	2020.07	29	2020.07		
10	2020.07	30	2020.07		
11	2020.07	31	2020.07		
12	2020.07	32	2020.07		
13	2020.07				
14	2020.07				
15	2020.07				
16	2020.07				
17	2020.07				
18	2020.07				
19	2020.07				
20	2020.07				

CAIRSENS PM

1 GENERAL – CHARACTERISTICS

The CAIRSENS PM consists of a storage box (1), a sensor for PM1, PM2.5 and PM10 (2) measurement, a black, 8 cm long, sampling tube (3).

(1) storage box, (2) sensor, (3) sampling tube

Figure 1-1 – Presentation of CAIRSENS PM in its storage box

The CAIRSENS PM sensor can be integrated into a CAIRNET mini-station which autonomously manages the acquisition and feedback of data (PM1, PM2.5, PM10 concentrations, temperature and relative humidity internal to the sensor) on the CAIRCLOUD.

The CAIRSENS PM sensor can also be used in a standalone version. It supports two communication protocols: UART and Modbus. The CAIRSENS PM which operates in UART protocol delivers a UART TTL 3V output signal.

WARNING: By default, the CAIRSENS PM operates in UART protocol (micro USB B port on the screen side).

For a standalone version use, it is necessary to use the request frames to be sent to the CAIRSENS PM in order to obtain the desired data:

- For UART protocol:

  Refer to paragraph « CAIRSPM Sensor » of the appendix document.

WARNING: – The UART protocol described in this document provides access to two measurements only: PM10 and PM2.5 concentrations. The other measurements, PM1, temperature and internal relative humidity of the sensor, are not accessible with the UART protocol described in this document.

– DO NOT connect the CAIRSENS PM directly to the USB port of a PC. Use a UART to USB converter (FTDI 3v3).

- For Modbus protocol:

  Refer to appendix document. This protocol enables access to the three concentrations measurements: PM10, PM2.5 and PM1, as well as the temperature and relative humidity measurements internal to the sensor.

The CAIRSENS PM can also operate with a specific software tool to download, configure, visualize and export data, named Cairsoft (new 2020 version). This tool is available for free in the "Download" section on our website www.cairpol.com. In this case too, a UART to USB converter cable must be used to connect the CAIRSENS PM to the PC (see appendix drawing).

1.1 GENERAL

1.1.1 PRESENTATION

The CAIRSENS PM sensor enables continuous and real-time measurement of the fine particulate concentrations PM1, PM2.5 and PM10 (in µg/m3) present in the ambient air. It performs measurements of these concentrations as an indication: the total uncertainty inferior to 50% is in compliance with the data quality objectives set by the European Directive 2008/50/EC on ambient air quality and clean air for Europe.

The CAIRSENS PM sensor is calibrated in a metrological qualification laboratory by ENVEA with a validity period of one year. It is compact and uses very few energy. It provides concentrations every minute.

The CAIRSENS PM sensor can be integrated into the CAIRNET mini-stations for several pollutant monitoring, or in a customized way into an air quality monitoring network.

The most usual applications of the CAIRSENS PM sensor are the following:

- Indoor and outdoor air quality monitoring: smart cities, roadsides and tunnels, schools, airports, ship terminals...
- Data providing for modelling atmospheric dispersion.
- Health and safety: mines, industrial sites, building.
- Emission forecasts at the industrial site boundaries.

1.1.2 DESCRIPTION

1.1.2.1 Front panel

See Figure 1-2.

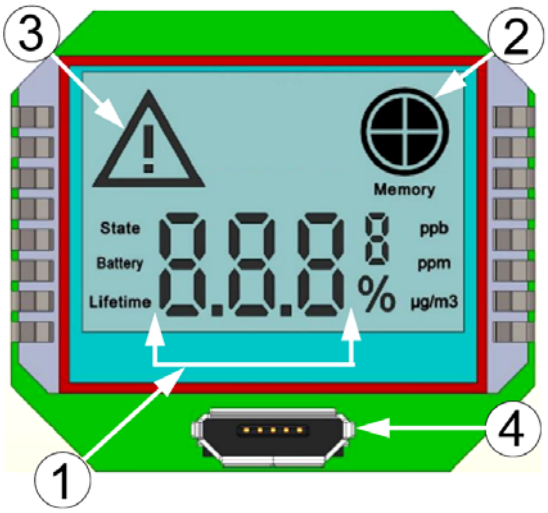
The front panel, protected by a polycarbonate plate, is fitted with:

– The LCD screen which displays :

- The measured concentration (1), represented with 3 digits and 1 exponent (see Chapter 2.).
- The status of the memory capacity used (2), by 25% fractions: when the whole memory is used, the circle quarters are black-filled.
- The possible CAIRSENS PM operation faults (3).

– The micro-USB B port (4), protected by a removable cap, which allows :

- The 5VDC CAIRSENS PM power supply,
- The UART CAIRPOL and UART MODBUS (slave mode) communication.



(1) measured concentration, (2) memory used, (3) operation errors, (4) micro USB B port

Figure 1-2 – CAIRSENS PM front panel

1.1.2.2 Sensor body

The sensor body includes the elements necessary for measurement. It is protected by a grey anodized aluminum external case EN AW-6060-T6 which holds the elements in place.

This CAIRSENS PM sensor does not include internal Lithium battery, but only a button cell maintains the timestamp.

The fluid circuit is very simple: the air to be analyzed is drawn-in through the sample inlet by a built-in micro fan, then it is exhausted through the hole located on the rear panel.

WARNING: DO NOT obstruct the sample gas inlet, nor its exhaust through the orifice located on the rear panel.

Figure 1-3 – Air exhaust orifice (1)

Figure 1-4 – Sample inlet view (2)

NOTE : Connect the sampling tube to the fitting (2) visible under the CAIRSENS PM body opening.

The following information is indicated on the CAIRSENS PM label: the pollutant measured (1), the measurement range (2), the supply voltage for connection (3), the CAIRSENS PM reference (4) and its serial number (5).

(1) Pollutant measured, (2) measurement range, (3) power supply,
(4) A40-0414 reference of CAIRSENS PM, (5) serial number

Figure 1-5 – CAIRSENS PM label

The sensor should not be eliminated in a traditional bin, but in an appropriate recovery and recycling structure.

1.1.3 OPERATING MODES

1.1.3.1 Standard

- Via the UART or MODBUS communication use: integrated in a customized solution with data centralization on a DAS for air quality supervision and monitoring, in association with other types of measurements (weather, noise, ...).
- Integration in a CAIRNET 3.0 mini-station

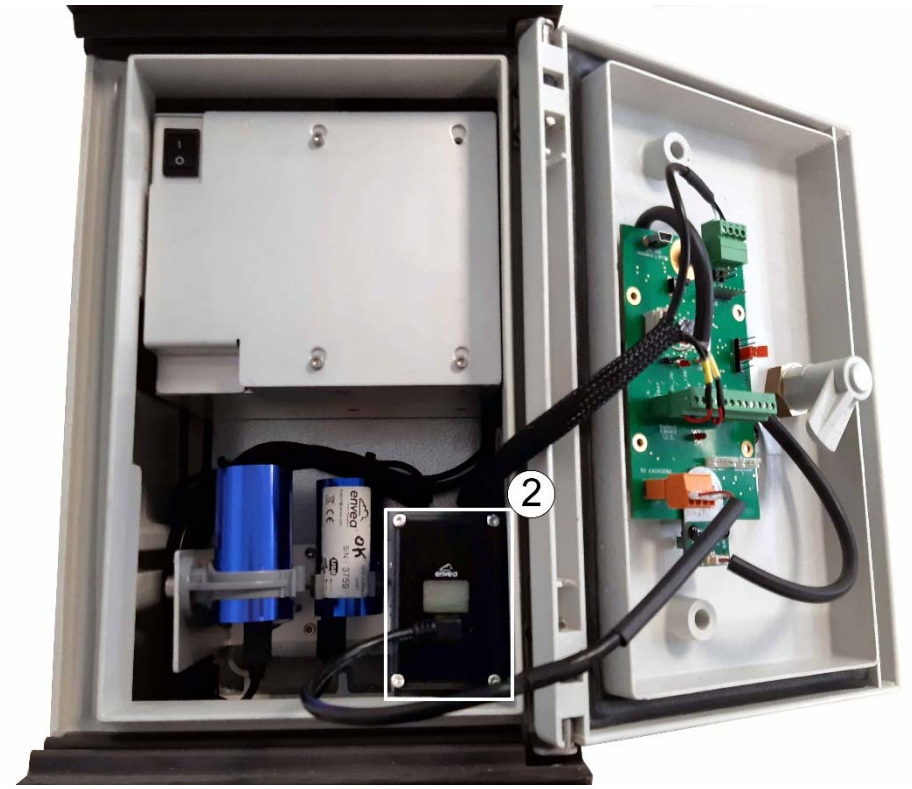
  The CAIRNET 3.0 mini-station can be power-supplied in two different ways:

  - Either with an 8-30 V compatible power supply,
  - Or autonomously by including the power supply option by rechargeable battery on photovoltaic panels and a 3G/4G cellular wireless communication module (Radio or GPRS), allowing a data feedback measurement on the CAIRCLOUD.

(1) CAIRSENS PM

Figure 1-6 – Integration in the CAIRNET 3.0 mini-station

NOTE : The CAIRSENS PM is compatible with the former CAIRNET 2.0 products.



(2) CAIRSENS PM

Figure 1-7 – Integration in a CAIRNET 2.0 mini-station

Note about the sampling tube length:

- For integration in a CAIRNET V3.0: the tube supplied with the sensor must be cut to have a length of 2.5 cm.
- For integration in a CAIRNET V2.0: the tube supplied with the sensor must be cut to have a length of 4 cm.

1.1.3.2 Option

The SAV-K-000290 kit contains all the parts required for integrating the CAIRSENS PM into a CAIRNET V2.0.

1.1.4 ASSOCIATED EQUIPMENT

The CAIRSENS PM can be associated to the following equipment (not supplied):

- Data Acquisition and Handling System

1.2 CHARACTERISTICS

1.2.1 TECHNICAL CHARACTERISTICS

The technical characteristics of the CAIRSENS PM are as follows:

Particulate matters measured	PM1, PM2.5 & PM10
Measurement range (3)	0 – 1 000 µg/m3
Detected particulate diameter (ø)	0.3 – 10 µm
Certified detection limit * (2)	< 5 µg/m3
Display resolution	0.01 µg/m3
Linearity (2)	R^2 > 0.75
Uncertainty between the sensors	< 5 µg/m3
Accuracy (slope) (2)	0.7 to1.3
Sample conditioning	Controlled air flow rate, Air heated over 60% of relative humidity
Temperature effect	< 0.01 µg/m3/°C
Technology	Laser light scattering
Operation temperature	-20 to 70 °C
Relative humidity of operation	0 to 95 HR % (without condensation)
Atmospheric pressure of operation	500 to 1 500 mbar

(2) Based on our laboratory assessment: daily PM2.5 average measurements compared to a reference.

(3) Arizona sand equivalent

The specifications for CAIRSENS PM integration in the client systems are:

CAIRSENS-PM sensor life time	1 year in continuous operation
Nominal power supply	– 5V DC / 500 mA, – USB port of a PC or an external battery (not supplied).
Electric consumption	250 mA max for 5 VDC
Gas sampling method	– Air flow controlled with a fan, – 2.5 L/min flow rate
Communications and inlet/outlet connection logins	UART Cairpol and UART Modbus (slave) via the micro-USB B port
LCD display	– Concentration in µg/m3, – Residual life time of the sensor – Operation status, – Available memory
Control and data processing board	Internal microprocessor for data acquisition and processing, integrated timer
Data storage capacity (internal)	– 2 days of 1-minute data – 30 days of 15-minutes data – or 120 days of 60-minutes data
Data download mode	– Customized integration / DAHS – CARNET mini-station (data export to CairCloud®) (option)

CAIRSENS PM compliance with environmental regulation:

Electrical safety	NF EN 61010-1: 2010
Electromagnetic compatibility	NF EN 61326-1: 2013
Protection index	IP 42 (according to CEI 60529)

1.2.2 STORAGE CHARACTERISTICS

The CAIRSENS PM storage conditions to be respected are:

Temperature (°C)	-20 to 70
Relative humidity (% HR)	0 to 95 (without condensation)
Pressure (mbar)	500 to 1500

1.2.3 INSTALLATION CHARACTERISTICS

1.2.3.1 Links between units

The CAIRSENS PM sensor uses the external links and supplies shown in the Figure 1-8 :

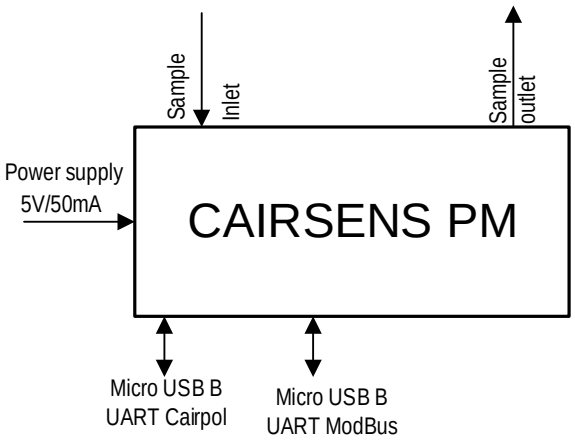


Figure 1-8 – Links between units
for CAIRSENS PM in standalone operation

1.2.3.2 Dimensions and weight

The device is in the form of a square block:

Length : 90 mm

Width : 45 mm

Height : 70 mm

Weight : 370 grams

1.2.3.3 Handling and storage

The CAIRSENS PM must be handled with precautions.

It should be stored in the box provided for this purpose.

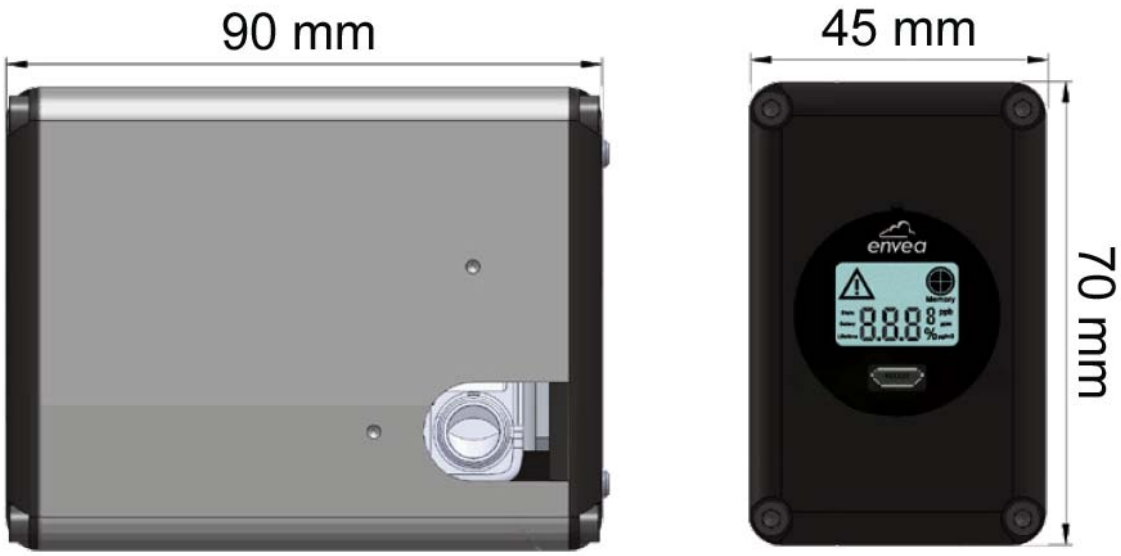


Figure 1-9 – CAIRSENS PM dimensions (in mm)

2 OPERATION PRINCIPLE AND MEASUREMENT

2.1 MEASUREMENT PRINCIPLE

The CAIRSENS PM performs measurements thanks to the presence of a particulate sensor able to measure simultaneously the PM10, PM2.5 and PM1 (µg/m3) concentrations.

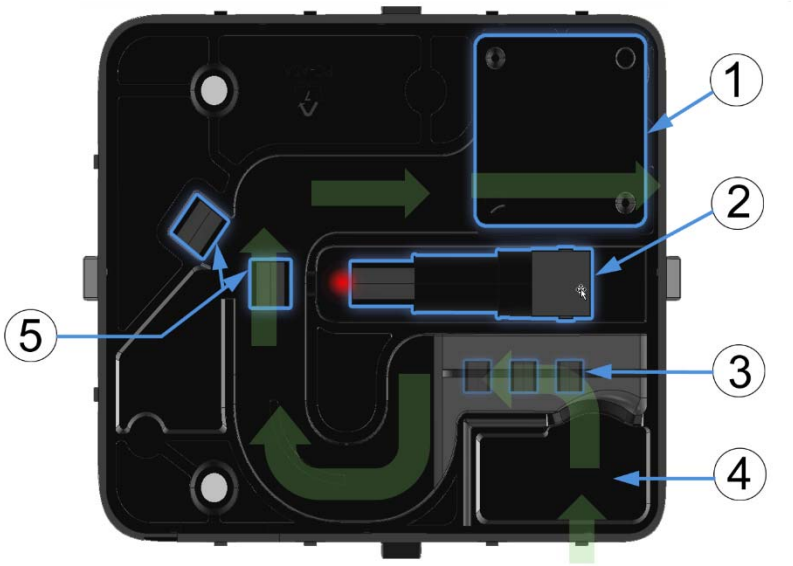
The particulates sampled in the measurement chamber pass through a continuous laser beam (2) and emit light in all directions according to the light scattering principle.

This scattered light is analyzed to calculate a mass concentration in µg/m3 for each size fraction: PM1, PM2.5 and PM10.

Air sampling is carried out using a controlled micro-fan (1) to maintain the flow rate at 2.5 L/min.

A sample heating system keeps the relative humidity measured in the measurement chamber below the 60% threshold, beyond which the optical particulate characteristics are altered and the sensor measurement is unreliable.

This heating system avoids the undesirable effects of a high relative humidity on the PM10, PM2.5 and PM1 measurement.



(1) Highly reliable managed micro fan, (2) high performance laser, (3) relative humidity and temperature management, (4) air inlet, (5) optical detectors.

Figure 2-1 – CAIRSENS PM measurement sensor

The graphs below give examples of results obtained with the sensor:

– The graph below shows the variation over time of PM2.5 concentrations (daily averages) simultaneously measured on a same measurement site by a CAIRSENS PM and a MP101M beta gauge (AMS certified US EPA and QAL1 EN16450).

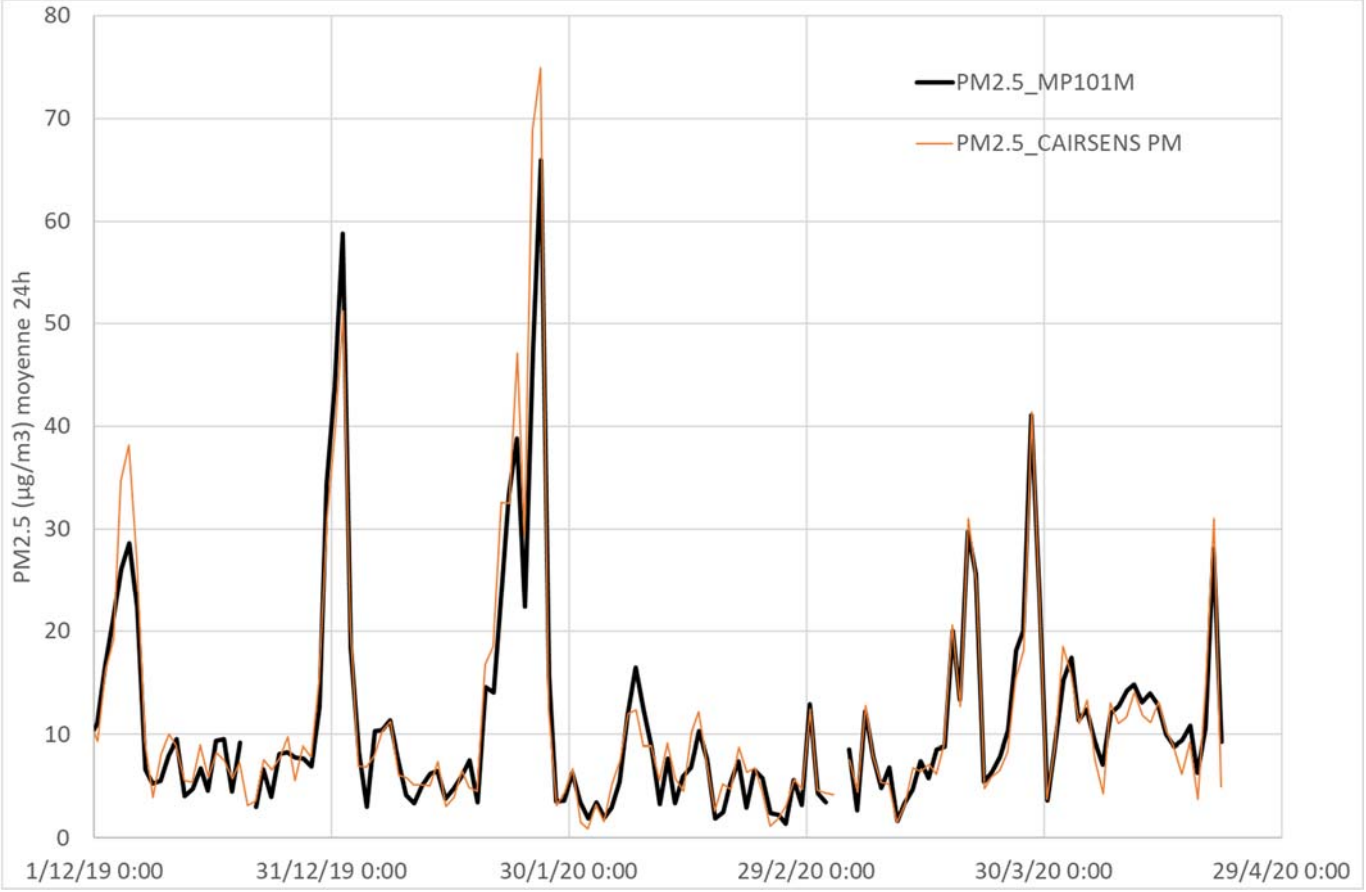


Figure 2-2 – Variation over time of PM2.5 concentrations

– The graph below compares the PM2.5 concentrations (daily averages) simultaneously measured on the same measurement site by a CAIRSENS PM and a MP101M beta gauge (AMS certified US EPA and QAL1 EN16450) from July 2019 to April 2020 (262 measurement days).

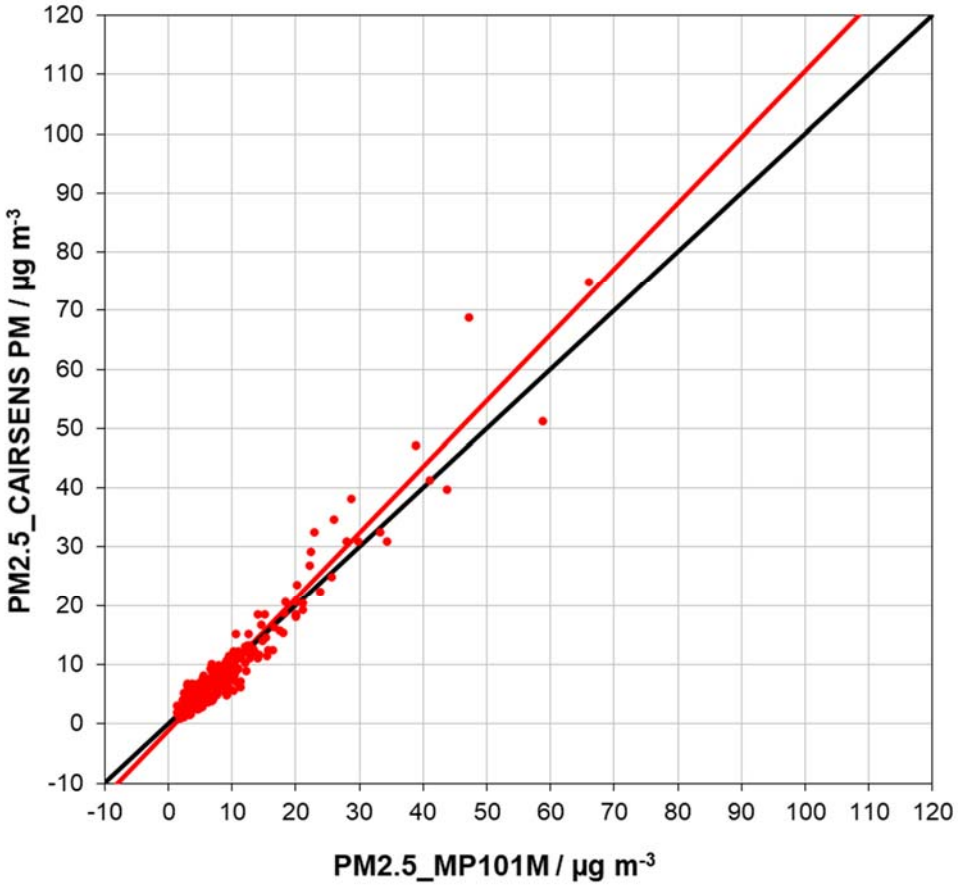


Figure 2-3 – Comparison of PM2.5 concentrations

NOTE : The resulting expanded uncertainty is 21.9% (the objective for the indicative measurement being 50%).

The slope and offset of the linear regression thus obtained are respectively equal to 1.1 and -1.1. These values are very satisfying for an indicative measurement.

2.2 MEASUREMENT

The CAIRSENS PM measurements present the following characteristics:

- Running averages of sensor measurements on 60 seconds. This average is recalculated each 10 seconds.
- Successive scrolling of PM10, PM2.5, PM1 measurements on screen.
- Measurement frequency stored in the CAIRSENS PM memory can be configured.

Concentration reading on the display:

See (1) Measured value displayed on three XXX digits, (2) « y » value in the formula: XXX ✖ 10y **Figure 2-4**.

The numerical value of the measured concentration is given by the formula:

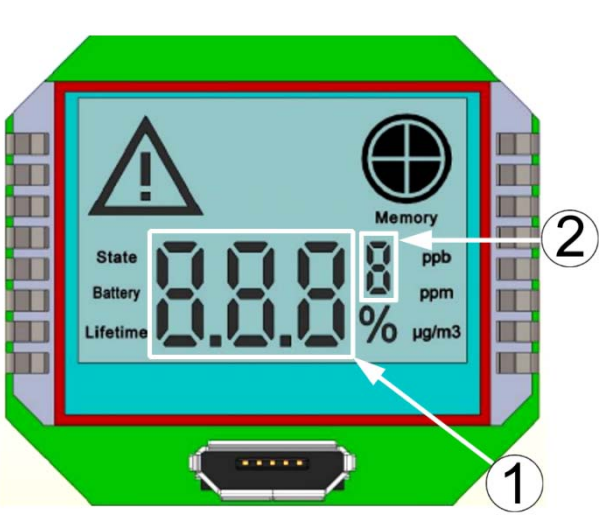
$$XXX \times 10^{y}$$

Where:

- XXX is the measurement value displayed on the three digits of the screen (1)
- ✖ is the multiplication operator
- 10^{y} is the multiplying coefficient to be applied to XXX. The “y” value is displayed in (2) on the screen.

As example, the concentration value for y = 0, y = 1 and y = 2 is given in the below table:

y value	10^{y} value	Concentration value reading
y = 0	$10^{0} = 1$	XXX ✖ 1 = XXX
y = 1	$10^{1} = 10$	XXX ✖ 10 = XXX0
y = 2	$10^{2} = 100$	XXX ✖ 100 = XXX00



(1) Measured value displayed on three XXX digits, (2) « y » value in the formula: XXX ✖ 10^{y}

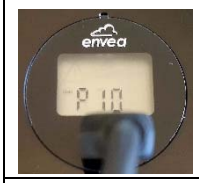
Figure 2-4 – Concentration display (in µg/m3)

As seen previously, the sensor simultaneously measures the PM10, PM2.5 and PM1 concentrations. The value display scrolls in identical sequence series on the screen, in the order shown below:

P10 ⇨ PM10 value only ⇨ P2.5 ⇨ PM2.5 value only ⇨ PM1 ⇨ PM1 value only.

When the sequence is finished, a new sequence starting with P10 scrolls.

Measurements are stated in µg/m3.

⇨ Measurement display sequence ⇨					
					
For PM10	[C] PM10 = 7,89 µg/m3	For PM2.5	[C] PM2.5 = 1,45 µg/m3	For PM1	[C] PM1 = 1,36 µg/m3

3 OPERATION

3.1 COMMISSIONING

3.1.1 INITIAL START-UP

CAIRSENS PM starts-up as soon as it is powered-up: CAIRSENS PM starts measurements as soon as it is running. The measurements are immediately displayed on the screen and stored automatically. The measurement backup is permanently performed in the CAIRSENS PM internal memory.

By default, the CAIRSENS PM is in continuous measurement mode, and the measurement period (or time step) is 1 minute. It can be modified using specific commands of the UART and Modbus protocols (see documentations in appendix) or by the CAIRSOFT software.

When the CAIRSENS PM memory is full, the sensor continues to operate normally, but records the new measurements by overwriting the oldest measurements.

When starting-up, all the pictograms are displayed by default (Figure 3-1), then the messages « init » and « notr » (NOT READY) are successively displayed on the screen:

Figure 3-1 – Display screens on CAIRSENS PM start-up

Remark: It is possible that the « fan » or « trh » alarms may be displayed on startup for a few seconds and then disappear automatically.

CAIRSENS PM calibration and measurement are guaranteed to be valid for 12 months after delivery.

CAIRSENS PM life time is 12 months (8760 h use). When this time is over, the screen display is: cAL (Figure 3-2). It is then necessary to replace the sensor by a new one.

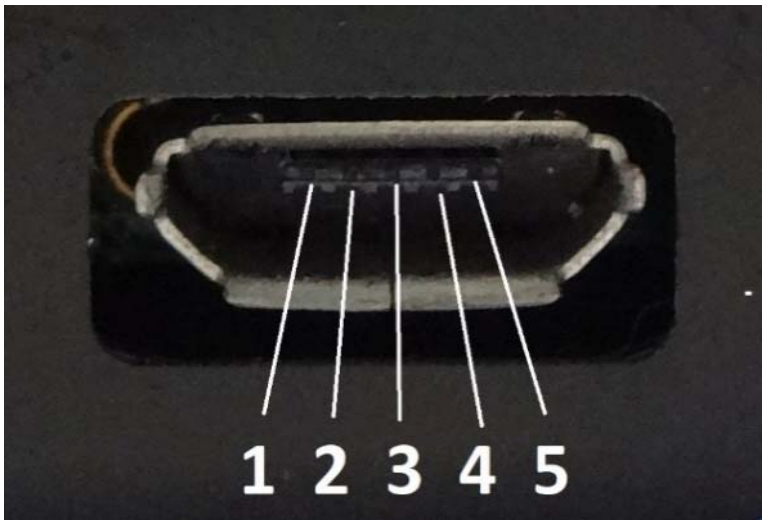
Figure 3-2 – Display screen at life time end

3.1.2 HOUR, DATE AND TIMESTAMP OF MEASURED DATA

The CAIRSENS PM internal clock is preset in factory, a button cell ensures the clock backup.

3.1.3 CAIRSENS PM WIRING FOR INTEGRATION

The wiring indication required to integrate CAIRSENS PM into a system is described below:



Pin Micro USB B	Description	Cable color
1	VDC +5V	Red
2	Data -	White
3	Data +	Green
4	Not used	Brown
5	GND Ground	Black

Figure 3-3 – Wiring indications for system integration

3.1.4 COMMUNICATION PROTOCOLS

The CAIRSENS PM supports two communication protocols, UART-CAIRPOL (by default) and UART-Modbus.

WARNING: By default, the CAIRSENS PM operates in UART protocol on the screen side (micro USB B port).

To change the communication protocol on the micro-USB port of CAIRSENS PM:

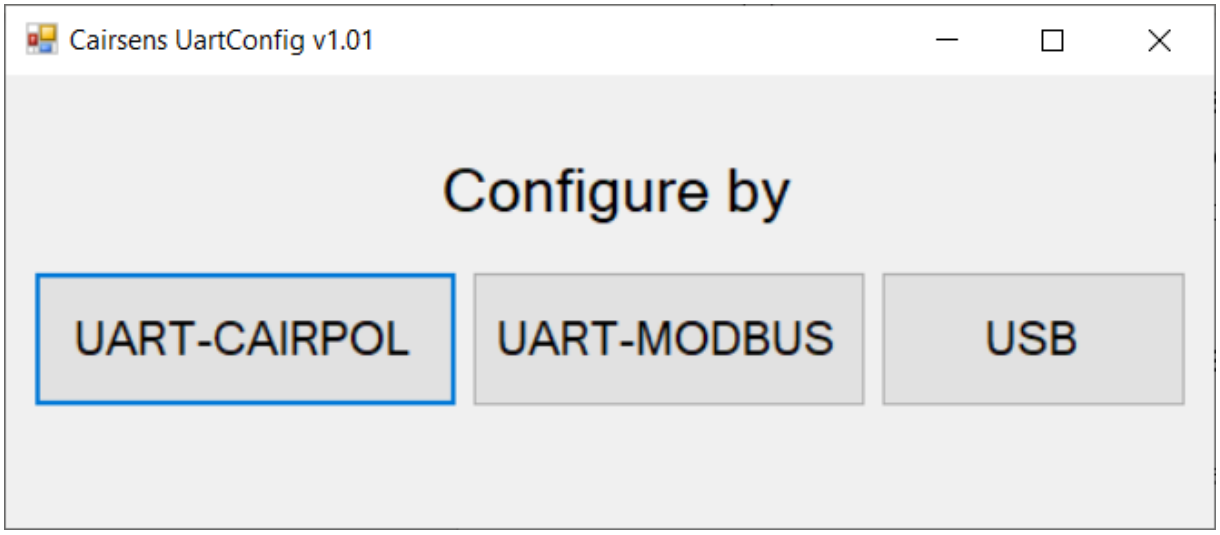
1. Connect the CAIRSENS PM to the PC using an UART to USB (FTDI 3v3) converter: see drawing in appendices.

It is advisable to use a fixed computer rather than a laptop PC to have a sufficient supply voltage for this CAIRSENS PM update.

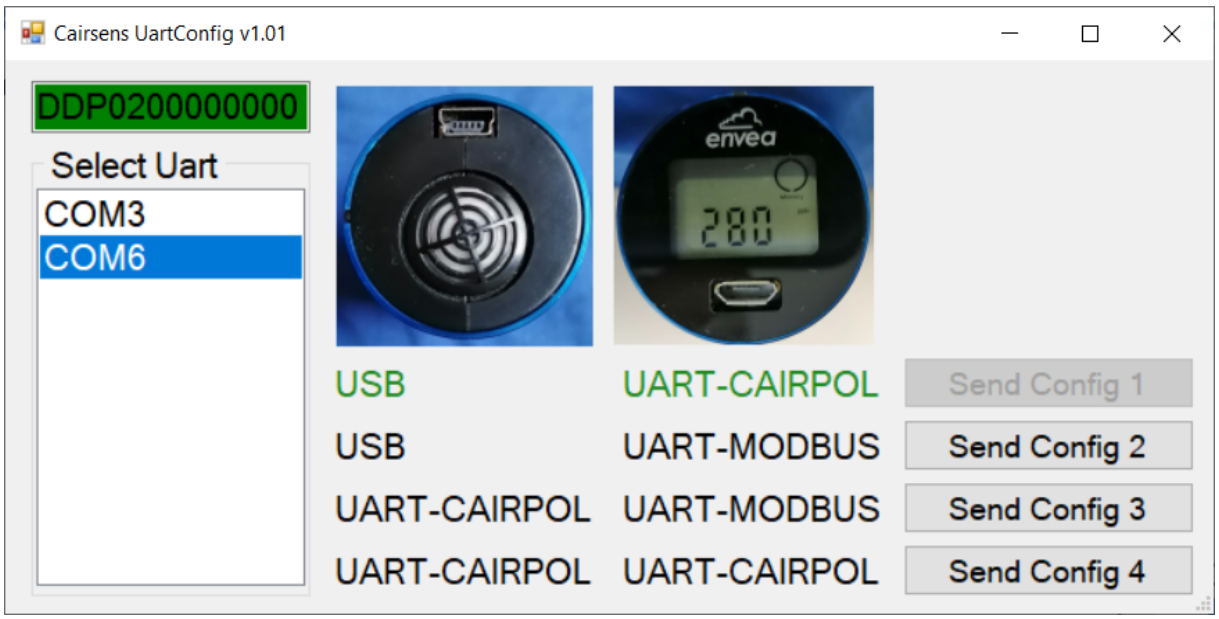
2. Execute the **uartconfig.exe** application freely downloadable from the web site « www.cairpol.com », at the same place than CAIRSOFT: « Download » heading, « Technical resource » tab.

3. In the window which opens when the application is started, select « UART-CAIRPOL » or « UART-MODBUS » depending on the initial configuration of the CAIRSENS PM.

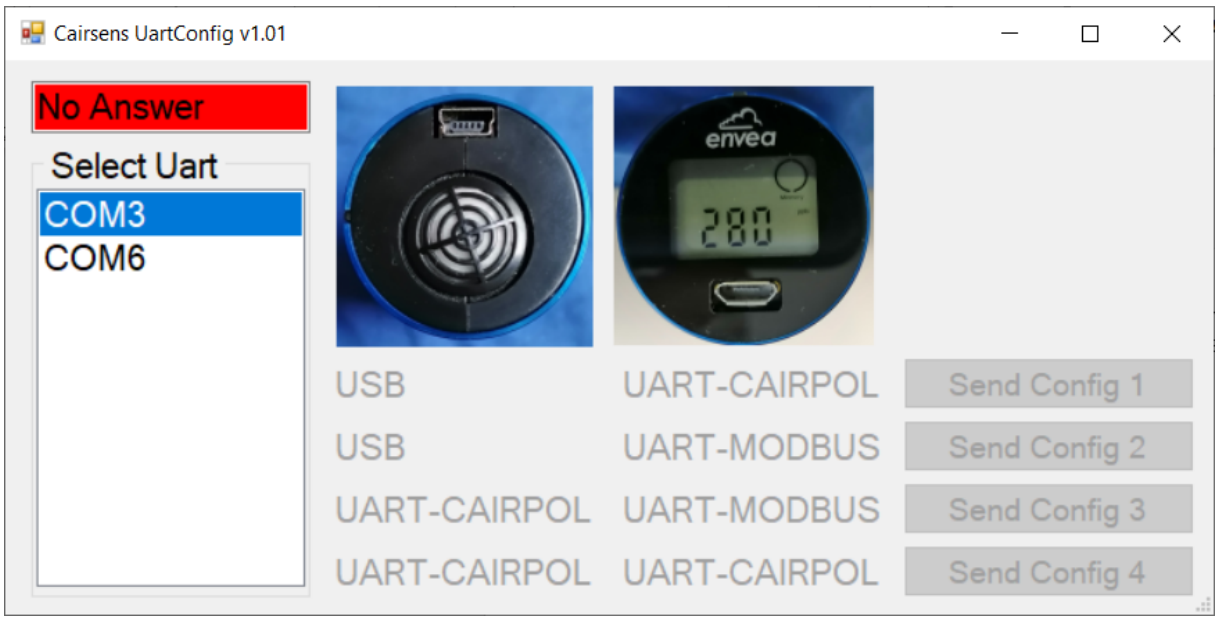
By default, CAIRSENS PM is configured with the UART CAIRPOL protocol when it leaves the factory. In this case, it is thus necessary to click on « UART-CAIRPOL »:



4. In the new window, select the COM port to which the CAIRSENS PM is connected. In the example below, click on « COM6 ». Once the sensor is detected, its serial number (in the form of DDP02XXXXXXXX) is displayed in the cell at the window left top and the cell turns green.

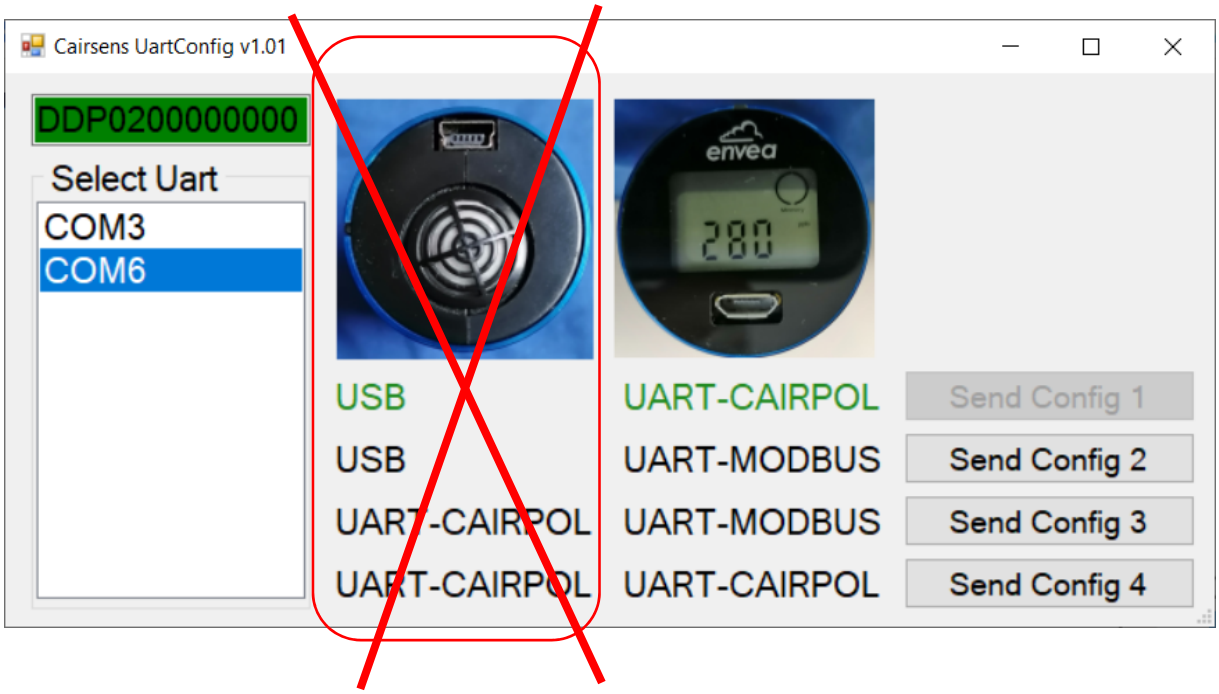


If the wrong COM port is selected or if the sensor is not detected, the cell at the window top left indicates « No Answer » and turns red:



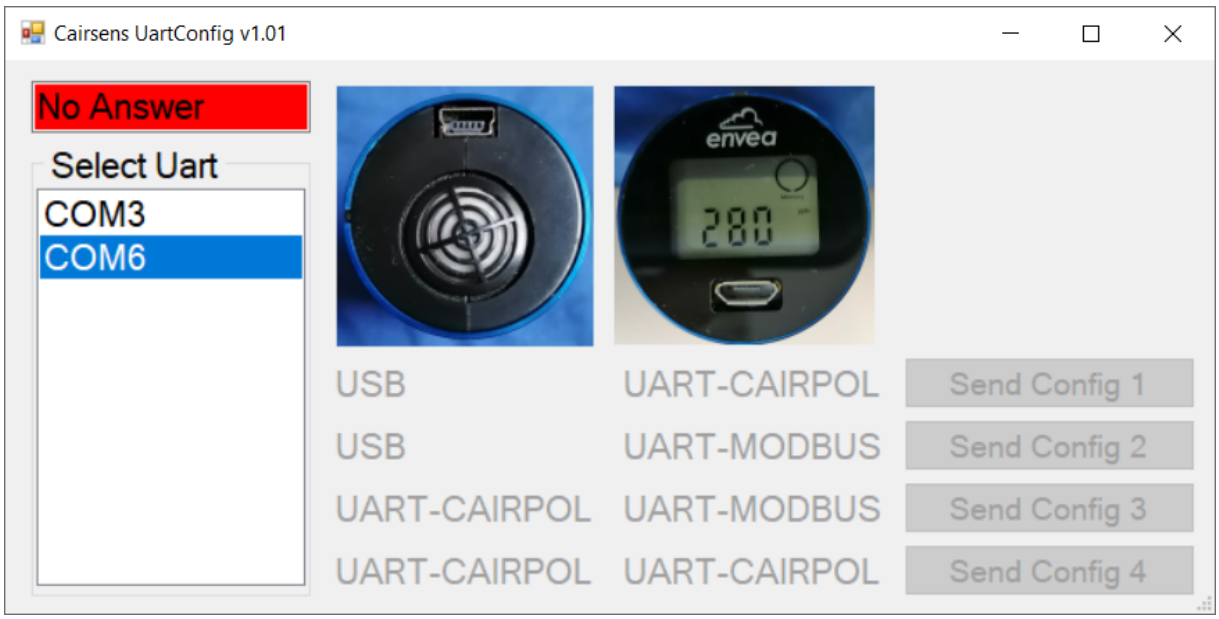
5. When the sensor is well detected, its current configuration is displayed in green on the right side of the screen (here: « UART-CAIRPOL » in the second column).

ATTENTION: For the CAIRSENS PM, only the second column under the screen picture should be considered. The first column concerns only the mini-USB port located behind the gas sensors of the CAIRSENS series.



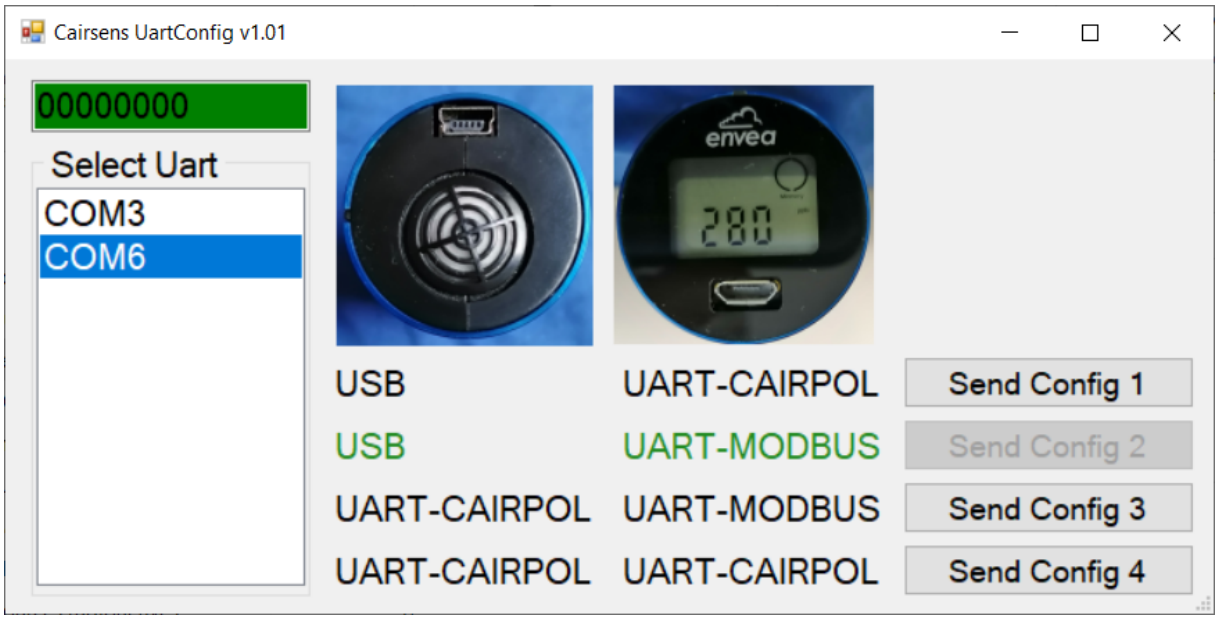
6. The configuration can then be changed. To do this, select the desired configuration and click on the button directly placed on its right. For example, in the present case, the switch from UART-CAIRPOL configuration to UART-MODBUS configuration is performed by clicking either on « Send Config 2 » or on « Send Config 3 ».

When the operation is finished, the cell at the window top left displays the message « No Answer » and turns red because the start configuration is no longer active.



7. To check that the configuration change occurred correctly, the following tests can be performed:

- Close the window
- Execute again uartconfig.exe
- In the first window, select the configuration that has just been loaded
- Select the COM port on which the sensor is connected
- Check that the sensor is well detected and configured as wished. In this present case, the configuration change from UART-CAIRPOL to UART-MODBUS ran well, and the UART-MODBUS configuration is well displayed in green in the right column :



4 OPERATION FAULTS

The detectable faults and displayed on the screen are:

On-screen message	Fault description
NOTR 	**NOT READY** The sensor is in warm-up. This display appears at start-up.
LASE	**LASER ERROR** Contact ENVEA by email to info@cairpol.com.
HEAT	**HEAT ERROR** The relative humidity remains greater than 60% for more than 10 minutes. The sensor can always send data, but they may be less precise.
TRH	**TRH ERROR** The sensor temperature and humidity measurements are out of specifications. The sensor can always send data, but they may be less precise. This error can appear during warm-up.
FAN	**FAN ERROR** The fan speed is out of tolerance but the fan is always running. The sensor can always send data, but they may be less precise. This error can appear during warm-up.
MEM	**MEMORY ERROR** The sensor cannot access its memory, some internal smart functions will not be available. The sensor can always send data, but they may be less precise.
SLEE	**SLEEP STATE** The laser, fan and heater are off.
CAL	**LIFE TIME END** The sensor has reached the end of its 1-year lifetime. After this period, the CAIRSENS PM metrological performances are no longer guaranteed. The laser may no longer work. It is recommended to renew the CAIRSENS PM quickly (contact ENVEA by email at info@cairpol.com).

On-screen message	Fault description
	**FULL MEMORY** The sensor continues to operate normally, but it records the new measurements by overwriting the oldest measurements (in the display shown here, it measures 0). The user must retrieve data regularly before reaching the maximum storage capacity of the CAIRSENS PM in order to avoid losing measurements.

5 CAIRSENS PM MAINTENANCE

5.1 SAFETY INSTRUCTIONS

The safety instructions must be observed at any time by the user.

- Switch off the power supply when servicing the CAIRSENS PM.
- Personnel must be properly trained to correct CAIRSENS PM operation before starting to operate it.
- Use only the provided accessories.
- Do not obstruct the air inlet nor the air extraction orifice.
- Do not hold the CAIRSENS PM in hands during the measurements.
- Do not use the CAIRSENS PM in a dusty, corrosive, explosive environment, and in the presence of other gases (combustion gases, solvent, chlorine, acid and basic vapors, etc.).
- Respect the conditions of use (see Technical characteristics).

Concerning safety, the manufacturer shall not be responsible for any adverse outcomes resulting from the followings:

- Use of the device by unqualified service personnel.
- Use of the device under conditions other than those specified in this document.
- Use of spare parts or accessories not manufactured by ENVEA.
- Use of this equipment in a manner not approved by ENVEA is not recommended and may result in damage to personnel and equipment. Failure to use specific spare parts may reduce the effectiveness of the safety device.
- The device modification by the user.
- No-maintenance of the device.

5.2 MAINTENANCE OPERATION

CAIRSENS PM is maintenance-free. It must be replaced annually.

6 APPENDICES

6.1 MODBUS PROTOCOL

6.2 CAIRSENS PM UART PROTOCOL

6.3 DRAWING

Page laissée blanche intentionnellement

Modbus RTU for CAIRSENS PM

1.0.1

Table of Contents

1 General

1.1 Overview

The official Modbus specification can be found at www.modbus.org/specs.php

Version	Date	Comments
1.0.0	July 2019	Initial manual

1.2 Available functions codes

Code	Action
03	Read holding registers
06	Write single holding register
16	Write multiple holding registers
23	Read and write multiples holding registers

2 MODBUS mapping

2.1 General parameters

Address	Register	Access	Data	Description
0 .. 9	1 .. 10	R	String	ENVEA
10 .. 19	11 .. 20	R	String	Version(ex 1.52)
20 .. 29	21 .. 30	R	String	Serial
30 .. 39	31 .. 40	R	String	Gaz (ex CO)
40	41	R/W	uint16	Year (ex 2019)
41	42	R/W	uint16	Month (1 to 12)
42	43	R/W	uint16	Day (1 to 31)
43	44	R/W	uint16	Hours (0 to 23)
44	45	R/W	uint16	Minutes (0 to 59)
45	46	R/W	uint16	Seconds (0 to 59)

2.2 Measure

Address	register	Access	Data	Description
80 .. 81	81 .. 82	R	float	PM10(µg/m3)
82 .. 83	83 .. 84	R	float	PM2.5(µg/m3)
84 .. 85	85 .. 86	R	float	Temp(°c)
86 .. 87	87 .. 88	R	float	Humidity(%)
88 .. 89	89 .. 90	R	float	PM1 (µg/m3)

2.3 Stored data PM10 (µg/m3)

Address	Register	Access	Data	Description
100 .. 101	101 .. 102	R	float	Last memorized value T0
102 .. 103	103 .. 104	R	float	memorized value T0- 1 minute
104 .. 105	105 .. 106	R	float	memorized value T0- 2 minutes

106 .. 107	107 .. 108	R	float	memorized value T0- 3 minutes
108 .. 109	109 .. 110	R	float	memorized value T0- 4 minutes
110 .. 111	111 .. 112	R	float	memorized value T0- 5 minute
112 .. 113	113 .. 114	R	float	memorized value T0- 6 minute
114 .. 115	115 .. 116	R	float	memorized value T0- 7 minute
116 .. 117	117 .. 118	R	float	memorized value T0- 8 minute
118 .. 119	119 .. 120	R	float	memorized value T0- 9 minute

2.4 Stored data PM2.5 (µg/m3)

Address	Register	Access	Data	Description
120 .. 121	121 .. 122	R	float	Last memorized value T0
122 .. 123	123 .. 124	R	float	memorized value T0- 1 minute
124 .. 125	125 .. 126	R	float	memorized value T0- 2 minutes
126 .. 127	127 .. 128	R	float	memorized value T0- 3 minutes
128 .. 129	129 .. 130	R	float	memorized value T0- 4 minutes
130 .. 131	131 .. 132	R	float	memorized value T0- 5 minute
132 .. 133	133 .. 134	R	float	memorized value T0- 6 minute
134 .. 135	135 .. 136	R	float	memorized value T0- 7 minute
136 .. 137	137 .. 138	R	float	memorized value T0- 8 minute
138 .. 139	139 .. 140	R	float	memorized value T0- 9 minute

2.5 Stored data Temp (°C)

Address	Register	Access	Data	Description
140 .. 141	141 .. 142	R	float	Last memorized value T0
142 .. 143	143 .. 144	R	float	memorized value T0- 1 minute
144 .. 145	145 .. 146	R	float	memorized value T0- 2 minutes
146 .. 147	147 .. 148	R	float	memorized value T0- 3 minutes
148 .. 149	149 .. 150	R	float	memorized value T0- 4 minutes
150 .. 151	151 .. 152	R	float	memorized value T0- 5 minute
152 .. 153	153 .. 154	R	float	memorized value T0- 6 minute
154 .. 155	155 .. 156	R	float	memorized value T0- 7 minute
156 .. 157	157 .. 158	R	float	memorized value T0- 8 minute
158 .. 159	159 .. 160	R	float	memorized value T0- 9 minute

2.6 Stored data Humidity (%)

Address	Register	Access	Data	Description
160 .. 161	161 .. 162	R	float	Last memorized value T0
162 .. 163	163 .. 164	R	float	memorized value T0- 1 minute
164 .. 165	165 .. 166	R	float	memorized value T0- 2 minutes
166 .. 167	167 .. 168	R	float	memorized value T0- 3 minutes
168 .. 169	169 .. 170	R	float	memorized value T0- 4 minutes
170 .. 171	171 .. 172	R	float	memorized value T0- 5 minute
172 .. 173	173 .. 174	R	float	memorized value T0- 6 minute
174 .. 175	175 .. 176	R	float	memorized value T0- 7 minute
176 .. 177	177 .. 178	R	float	memorized value T0- 8 minute
178 .. 179	179 .. 180	R	float	memorized value T0- 9 minute

2.7 Stored data PM1 (ug/m3)

Address	Register	Access	Data	Description
180 .. 181	181 .. 182	R	float	Last memorized value T0
182 .. 183	183 .. 184	R	float	memorized value T0- 1 minute
184 .. 185	185 .. 186	R	float	memorized value T0- 2 minutes
186 .. 187	187 .. 188	R	float	memorized value T0- 3 minutes
188 .. 189	189 .. 190	R	float	memorized value T0- 4 minutes
190 .. 191	191 .. 192	R	float	memorized value T0- 5 minute
192 .. 193	193 .. 194	R	float	memorized value T0- 6 minute
194 .. 195	195 .. 196	R	float	memorized value T0- 7 minute
196 .. 197	197 .. 198	R	float	memorized value T0- 8 minute
198 .. 199	199 .. 200	R	float	memorized value T0- 9 minute

3 Appendix

3.1 Glossary

Address	address of location in memory map (WORD format => 2 bytes)
R	Read only parameter
R/W	Read / Write parameter
string	Character string
float	32 bits floating point BIGENDIAN format
register	Word of 16 bits

Cairsens - UART Version

Communication Protocol

Measured data download

Table of Contents

1 UART Port settings

Baud rate	data bits	Parity	Stop bits	Flow control
9600	8	N	1	none

2 Queries / answers structures between UART cairsens and host

The structure of the query / answer frame passing between the Cairsens and the host can be defined by a series of bytes, the number of which varying and being represented in hexadecimal.

The query frames have a fixed length and are structured as follows:
SYNC STX LG REF DATA CRC ETX

The answer frames have a fixed length and are structured as follows:
SYNC STX LG REF DATA END CRC ETX

Bytes definition is:
- SYNC = Synchro Word
- STX = Start Frame
- LG = Length of Data
- REF = Cairsens identification 7
- DATA = [CMD+PARAM] (Series of bytes for command and parameters)
- END =End Frame
- CRC [2 bytes/LSB First]
- ETX
- LIFE = Life used 4

Synchronization and start frame bytes have the following values and constant number of bytes:
- SYNC = 1 byte = 0xFF
- STX = 1 byte = 0x02
- CRC = 2 bytes
- END = 2 bytes = LIFE 0xFF
- ETX = 1 byte = 0x03

3 Life

Byte value	
0x00	the sensor can't return it's % life used
0x80	New sensor
0xC0	50 % of life used
0xE0	75% life used
0xFF	100% life used (end of life)

4 Available commands

CMD	Description
0x0C	Get stored data

0x12	Get the last minute average data (1 data returned)
0x1C	Get stored average data (multiple data returned)

5 HeaderUart and TrailerUart definitions

In the following section of this document, the series of bytes representing SYNC STX LG and a part of CMD will be referred to as **HeaderUart** and will be defined by:

- HeaderUart = SYNC, STX, LG, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06

In the same idea, the series of bytes representing END will be referred to as **TrailerUart** and will be defined by:

- TrailerUart = END CRC ETX

6 Cyclic redundancy checks

6.1 Compute

The CRC code is calculated by dividing the binary sequence representing the frame by the following polynomial:
$X^{16} + X^{12} + X^5 + 1$

6.2 Sample in c

```
#include <stdio.h>

unsigned int FCRC( unsigned char Frame[],unsigned char lg)
{
  unsigned int Poly = 0x8408;
  unsigned int Crc;
  unsigned char j,i_bits,carry;
  Crc=0;
  for (j=0;j<lg; j++) {
    Crc = Crc ^ Frame[j];
    for ( i_bits = 0; i_bits < 8; i_bits++ ) {
      carry = Crc & 1;
      Crc = Crc/2;
      if(carry) {
        Crc = Crc ^ Poly;
      }
    }
  }
  return Crc;
}

int main(int argc, char* argv[])
{
    unsigned int i;

    unsigned char Frame[] = {0xFF, // Synchro Word
    0x02,// Start Frame
    0x13,// Length of Data
     0x30,0x01,0x02,0x03,0x04,0x05,0x06,
```

```
    0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
    0x12, // CMD
    0x00,0x00, // CRC [2 bytes/LSB First]
    0x03}; // End Frame

    unsigned int StartPos = 2; // start position CRC

    printf ( " Frame without CRC =" );
    for( i = 0 ; i < sizeof( Frame) ; i++ )
    {
        if( i > 0 ) putchar(',');
        printf ( " 0x%02X" , Frame[i] );
    }
    putchar('\n');

    i = FCRC ( &Frame[StartPos] , Frame[StartPos] - 2); // compute CRC without
CRC's bytes

    printf ( " CRC=0x%04X\n" , i );

    Frame[19] = i & 0xFF;
    Frame[20] = i >> 8;
    printf ( " CRC IN FRAME(LSB First)= 0x%02X 0x%02X\n" , Frame[19] ,
Frame[20]);


    printf ( " Frame with CRC= " );
    for( i = 0 ; i < sizeof( Frame) ; i++ )
    {
        if( i > 0 ) putchar(',');
        printf ( " 0x%02X" , Frame[i] );
    }
    putchar('\n');


    i = FCRC ( &Frame[StartPos] , Frame[StartPos] );  // check CRC
    if( i == 0 )
       printf ( " CRC OK\n");
    else
       printf ( " CRC ERROR\n" );
}

// output
//
//
// Frame without CRC = 0xFF, 0x02, 0x13, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05,
0x06,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x12, 0x00, 0x00, 0x03
// CRC=0x88AF

// CRC IN FRAME(LSB First)= 0xAF 0x88

// Frame with CRC=  0xFF, 0x02, 0x13, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x12, 0xAF, 0x88, 0x03       //
CRC OK
```

7 REF definition

7.1 Product option

The REF 8 bytes represents the Cairsens reference.
It allows to address individually and directly to a Cairsens, when in a network, several Cairsens are linked to a unique concentrator card.
The reference is included in every query with the Cairsens to allow an individual addressing, as only the concerned Cairsens will answer to the query.

FF FF FF FF FF FF FF FF is a generic address allowing to communicate with any product without knowing its reference.
of course in this situation, it has to be used with only one Cairsens linked to the host, to avoid any BUS corruption as all sensors will respond.

First byte is the product ID :

C = CAIRCLIP
D = CAIRSPM **(new for particulates data and battery management)**
H = CairClip H2S 200ppm
M = CairClip H2S 20ppm
L = CairClip H2S 2ppm

The reference is an 8 bytes series coded as follows: XX YY ZZ AA 00 00 00 00

XX	Product ID C=0x43 D=0x44
YY	Gas identification
ZZ	Measure Range
AA	Interface Type
00 00 00 00	serial number

7.2 Coefficient

For each sensor, you must use a multiplicative coefficient to get the final value :

Sensor	Code	coefficient
CO 20ppm	COV	1
NMVOC 16ppm	CIV	1
H2S 1ppm	CHM	4
H2S 200ppm	CHV	10
H2S 20ppm	CHV or HHV	1
H2S 2ppm	CHV or MHV	1
NH3 25ppm	CAV or LHV	100
O3 1ppm	CCM	4
O3 250ppb	CCB	1
SO2 1ppm	CSM	4

7.3 Gaz Identification

Product reference second byte gives the gas identification (NH3 in the example below)

REF	8 bytes	0x43
		0x41
		0x56
		0x32
		0x39
		0x44
		0x30
		0x35

List of gases

ASCII	HEX	DATA
A	**0x41**	Ammonia(NH3)
B	0x42	Benzene
C	0x43	Ozone(O3) and Nitrogen Dioxide (NO2)
D	0x44	Dust
E	0x45	CO2
F	0x46	Formaldehyde(CH2O)
G	0x47	CH4
H	0x48	Hydrogen Sulfide(H2S)
I	0x49	NMVOC
L	0x4A	Chlorine(Cl2)
N	0x4B	Nitrogen Dioxide(NO2)
O	0x4C	CO
P	0x4D	Tetrachloroethylene
T	0x4E	Toluene
S	0x4F	SO2

7.4 Measure range

ASCII	HEX	Range
B	0x42	0-250 ppb
M	0x4D	0-1 ppm
V	0x56	0-20 ppm
P	0x50	PACKET data block for CAISPM

7.5 Interface type

HEX	Interface
0x01	USB

0x02	UART

8 Reading of the instant value of the Cairsens (GetValue)

8.1 Query

This structure allows the reading of the instant value of the Cairsens (last 1minute data stored)
Query:
- HeaderUart REF CMD CRC ETX
- Command byte CMD = 0x12

8.2 Answer

HeaderUart REF RSP PARAM=0xXX CRC TrailerUart
- Answer byte RSP = 0x13
- Instant value byte PARAM (see below)

The last value (last data stored) is expressed as follows:
- Parameter 1: 1 data

8.3 Example 1 byte by value

- Query:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x13
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x12**
CRC	2 bytes	0xAF
		0x88
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x16
	7 bytes	0x2C
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0x43
		0x41
		0x56
		0x32
		0x39
		0x44
		0x30
		0x35
RSP	1 byte	**0x13**
Measure	1 byte	0xD1
END	2 byte	0x00
		0xFF
CRC	2 bytes	0x70
		0xFB
ETX	1 byte	0x03

here the value is 0xD1 = 209

it's a cairsens CAV (NH3 25ppm) : measure = 209*100 = 20900pbb = 20.9 ppm

for a cairsens CHM (H2S 1ppm) : measure = 209*4 = 836pbb = 0.836 ppm
for a cairsens CCM (O3 1ppm) : measure = 209*4 = 836pbb = 0.836 ppm
for a cairsens CSM (SO2 1ppm) : measure = 209*4 = 836pbb = 0.836 ppm

8.4 Example 2 bytes by value

- Query:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x13
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x12**
CRC	2 bytes	0xAF
		0x88
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x17
	7 bytes	0x2C
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0x43
		0x49
		0x56
		0x32
		0x33
		0x33
		0x30
		0x33
RSP	1 byte	**0x13**
	2 bytes	0xB8
		0x2E
END	2 byte	0x00
		0xFF
CRC	2 bytes	0xF3
		0x8D
ETX	1 byte	0x03

it's a cairsens CIV (2 bytes by value)
measure = (0x2E * 256 + 0xB8) = 11960 ppb = 11.960 ppm

for a cairsens H2S 200ppm the value is: 11960 * 10 = 119600ppb = 119.6 ppm

9 GetDownload structure for cairsens (Stored data download)

9.1 Query

Command byte is **CMD** = 0x0C

The parameter allowing the data download **PARAM** is built on a byte which value can vary and refer to several periods to download:

- 0x00: 10 successive points of measurement
- 0x01: 96 successive points of measurement for 1 byte by value , 48 successive points of

measurement for 2 byte by value
- 0x02: send 7 answer of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value
- 0x03: send 30 answers of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value
- 0x04: send 60 answers of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value
- 0x05: send 90 answers of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value
- 0x06: send 240 answers of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value
- 0x07: send 300 answers of 96 successive points of measurement for 1 byte by value , 48 successive points of measurement for 2 byte by value

This number of points of measurement is valid for a sampling factory configured (meaning one measurement per minute)

9.2 Answer

HeaderUart REF RSP PARAM **TrailerUart**

Answer byte is RSP = 0x0D
- Information + requested data = PARAM (see below)

PARAM holds various information about Cairsens' status in addition to the requested data.

This sequence of information consists of the 10 following parameters:
- Parameter 1 - 1 byte: number of the RS232 exchange frame, coded in hexadecimal (from 0x01 to 0xFF)
- Parameter 2 - 1 byte: total number of RS232 exchange frames, coded in hexadecimal (from 0x01 to 0xFF)
- Parameter 3 - 2 bytes: not use
- Parameter 4 - 1 byte: not use
- Parameter 5 – 1 byte: not use
- Parameter 6 – 1 byte: not use
- Parameter 7 – 1 byte: not use
- Parameter 8 – 1 byte: not use
- Parameter 9 – 2 bytes: not use
- Parameter 10 – 96 bytes:

  1 byte by value : 96 data of 1 byte each = 96 bytes of pollutant level data
  2 bytes by value : 48 data of 2 byte each = 96 bytes of pollutant level data

9.3 Example 10 minutes data 1 byte by value

- Query:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x14
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x0C**
PARAM	1 byte	0x00
CRC	2 bytes	0x63
		0xA8
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x2A
	7 bytes	0x2C
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0x43
		0x48
		0x4D
		0x02
		0x09
		0x14
		0x00
		0x22
RSP	1 byte	**0x0D**
number of the RS232 exchange frame, coded in hexadecimal	1 byte	0x01
total number of RS232 exchange frames, coded in hexadecimal	1 byte	0x01
year on which data storage began (from 0x0000 to 0x2399) LSB first BCD	2 bytes	0x00
		0x00
month on which data storage began (from 0x00 to 0x11) with January = 0x00 - BCD	1 byte	0x00
day on which data storage began (from 0x01 to 0x31) - BCD	1 byte	0x00
hour on which data storage began (from 0x00 to 0x12) - BCD	1 byte	0x00
minutes on which data storage began (from 0x00 to 0x59) - BCD	1 byte	0x00
AM/PM flag (AM = 0x00, PM = 0x01) – BCD	1 byte	0x00
running meter in Cairsens' memory (from 0x0000 to 0x7BC0) LSB first HEX	2 bytes	0x6F
		0x1B
value n°1(oldest)	1 byte	0x00
value n°2	1 byte	0x00
value n°3	1 byte	0x00
value n°4	1 byte	0x00
value n°5	1 byte	0x00
value n°6	1 byte	0x00
value n°7	1 byte	0x00
value n°8	1 byte	0x00
value n°9	1 byte	0x00
value n°10(recent)	1 byte	0x00
END	2 bytes	0x00
		0xFF
CRC	2 bytes	0x4D
		0x90
ETX	1 byte	0x03

for a cairsens CAV (NH3 25ppm) : measure in ppb = value*100
for a cairsens CHM (H2S 1ppm) : measure in ppb = value*4
for a cairsens CCM (O3 1ppm) : measure in ppb = value*4
for a cairsens CSM (SO2 1ppm) : measure in ppb = value*4

9.4 Example 10 minutes data 2 bytes by value

Download of 10 minutes data => GetDownload query (0x00):

- Query:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x14
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x0C**
PARAM	1 byte	0x00
CRC	2 bytes	0x63
		0xA8
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x34
	7 bytes	0x2C
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	0x43
		0x49
		0x56
		0x02
		0x33
		0x33
		0x00
		0x33
RSP	1 byte	**0x0D**
number of the RS232 exchange frame, coded in hexadecimal	1 byte	0x01
total number of RS232 exchange frames, coded in hexadecimal	1 byte	0x01
year on which data storage began (from 0x0000 to 0x2399) LSB first BCD	2 bytes	0x00
		0x00
month on which data storage began (from 0x00 to 0x11) with January = 0x00 - BCD	1 byte	0x00
day on which data storage began (from 0x01 to 0x31) - BCD	1 byte	0x00
hour on which data storage began (from 0x00 to 0x12) - BCD	1 byte	0x00
minutes on which data storage began (from 0x00 to 0x59) - BCD	1 byte	0x00
AM/PM flag (AM = 0x00, PM = 0x01) – BCD	1 byte	0x00
running meter in Cairsens' memory (from 0x0000 to 0x7BC0) LSB first HEX	2 bytes	0xFE
		0x0F
value n°1(oldest)	2 bytes	0xE8
		0x2B
value n°2	2 bytes	0x60
		0x2C
value n°3	2 bytes	0x1A
		0x2C
value n°4	2 bytes	0x8E
		0x2B
value n°5	2 bytes	0x8E
		0x2B
value n°6	2 bytes	0x8E
		0x2B
value n°7	2 bytes	0x06

		0x2C
value n°8	2 bytes	0x60
		0x2C
value n°9	2 bytes	0xDE
		0x2B
value n°10(recent)	2 bytes	0xE8
		0x2B
END	2 bytes	0x00
		0xFF
CRC	2 bytes	0x69
		0x0D
ETX	1 byte	0x03

value 1 : 0x2B *256 + 0xE8 = 11240 ppb
value 2 : 0x2C *256 + 0x60 = 11360 ppb
value 3 : 0x2C *256 + 0x1A = 11290 ppb
value 4 : 0x2B *256 + 0x8E = 11150 ppb
value 5 : 0x2B *256 + 0x8E = 11150 ppb
value 6 : 0x2B *256 + 0x8E = 11150 ppb
value 7 : 0x2C *256 + 0x06 = 11270 ppb
value 8 : 0x2C *256 + 0x60 = 11360 ppb
value 9 : 0x2B *256 + 0xDE = 11230 ppb
value 10: 0x2B *256 + 0xE8 = 11240 ppb

for a cairsens H2S 200ppm : measure 1 = value1*10 = 112400 ppb = 112.4 ppm

9.5 Acquisition example for windows in c

```
#include <stdio.h>
#include <time.h>

#include <windows.h>

typedef int bool;
#define false 0
#define true 1

unsigned char TrameInst[]= {0xFF, 0x02, 0x13, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x12, 0xAF, 0x88, 0x03};
unsigned char Trame10M[] = {0xFF, 0x02, 0x14, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x0FF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x0C, 0x00, 0x63, 0xA8, 0x03};


void* rs_open(const char *name)
{
        HANDLE h;
        h = CreateFileA(name,
                GENERIC_READ | GENERIC_WRITE,
                0, // must be opened with exclusive-access
                NULL, // default security attributes
                OPEN_EXISTING, // must use OPEN_EXISTING
                0, // not overlapped I/O
                NULL); // hTemplate must be NULL for comm devices

        if(INVALID_HANDLE_VALUE == h)
                return 0;

        return (void*)h;
}
```

```
int rs_close(void* port)
{
	HANDLE h = (HANDLE)port;
	CloseHandle(h);
	return 0;
}

int rs_setattr(void* port)
{
	DCB dcb;
	HANDLE h = (HANDLE)port;

	ZeroMemory(&dcb, sizeof(dcb));
	dcb.DCBlength = sizeof(dcb);
	GetCommState(h, &dcb);

	dcb.fBinary = TRUE;

	dcb.BaudRate = CBR_9600;

	dcb.ByteSize = 8;
	dcb.Parity = NOPARITY;
	dcb.StopBits = ONESTOPBIT;
	//Setup the flow control
	dcb.fDsrSensitivity = FALSE;
	dcb.fOutxCtsFlow = FALSE;
	dcb.fOutxDsrFlow = FALSE;
	dcb.fOutX = FALSE;
	dcb.fInX = FALSE;

	if(!SetCommState(h, &dcb))
		return -1;
	return 0;
}

int rs_write(void* port, const void* data, int bytes)
{
	DWORD n = 0;
	HANDLE h = (HANDLE)port;

	if(!WriteFile(h, data, bytes, &n, NULL))
		return -1;

	return n;
}

int rs_read(void* port, void* data, int bytes)
{
	DWORD n = 0;
	HANDLE h = (HANDLE)port;

	if(!ReadFile(h, data, bytes, &n, NULL))
		return -1;

	return n;
}


int rs_flush(void* port)
{
 HANDLE h = (HANDLE)port;
 FlushFileBuffers(h);
 return 0;
}


int rs_bytesWaiting(void* port)
{
	//Check to see how many characters are unread
	COMSTAT stat;
	DWORD dwErrors = 0;
	HANDLE h = (HANDLE)port;

	if (!ClearCommError(h, &dwErrors, &stat))
	{
```

```
                return 0;
        }
        return stat.cbInQue;
}


int readFrame( void * port , unsigned char * data)
{
        int nb;
        int i;
        int timeout;
        int iLen;
        int maxtimeout = 20;
        int waittempo = 100;
        int indexFrame;

        do {
                i = 0;
                indexFrame = 0;

                data[0] = 0;
                i = 1;
                timeout = 0;
                // Detect 0xFF
                do {
                        if( rs_bytesWaiting( port) > 0)
                        {
                                rs_read(port , &data[0] , 1);
                        }
                        Sleep(waittempo);
                        if( timeout > maxtimeout )
                        {
                                return 0;
                        }
                        timeout += 1;
                        // break loop if data(0) is 0xFF
                        if( 0xFF == data[0] ) i = 0;
                }while ( i > 0 );

                // detect 0x02
                timeout = 0;
                i = 1;
                do
                {
                        if( rs_bytesWaiting( port) > 0)
                        {
                                rs_read(port , &data[1], 1);
                        }
                        Sleep(waittempo);
                        if( timeout > maxtimeout )
                        {
                                return 0;
                        }
                        timeout += 1;
                        if( 0x02 == data[1])
                        { // 0x02 detected
                                i = 0;
                        }
                        indexFrame += 1;
                        if( indexFrame > (256 - 1) )
                        { //0x02 not detected
                                i = 0;
                                data[1] = 0;
                        }
                }
                while (i > 0);

                // 0x02 detected
                if( 0x02 == data[1] )
                {
                        timeout = 0;
                        do
                        {
                                i = rs_bytesWaiting( port);
                                Sleep(waittempo);
```

```
                                if( timeout > maxtimeout )
                                {
                                        return 0;
                                }
                                timeout += 1;
                        }while (i < 1);

                        i = rs_read(port , &data[2], 1);

                        iLen = data[2];

                        timeout = 0;
                        do
                        {
                                i = rs_bytesWaiting( port);
                                Sleep(waittempo);
                                if( timeout > maxtimeout )
                                {
                                        return 0;
                                }
                                timeout += 1;
                        } while (i < iLen);

                        i = rs_read(port , &data[3], iLen );
                        nb = i + 3;
                        return nb;
                }

        }while ( rs_bytesWaiting( port) > 0);

        return 0;
}


unsigned int CalculCrc( unsigned char *data, unsigned int start , unsigned int lg)
{
        unsigned int crc = 0;
        const unsigned int Poly = 0x8408;
        unsigned int  j, val, carry, i_bits;

        crc = 0;
        for( j = 0  ;  j < lg ; j++ )
        {
                val = data[j + start];
                crc = crc ^ val;
                for( i_bits = 0 ; i_bits < 8 ; i_bits++ )
                {
                        carry = crc & 1;
                        crc >>= 1;
                        if ( carry > 0 )   crc = crc ^ Poly;
                }

        }
        return crc;
}

void print_date( void )
{
  time_t rawtime;
  struct tm * timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  printf ( "%s", asctime (timeinfo) );
}

void read_val_inst( void * port )
{
        int i,nb;
        unsigned char data[1024];
        unsigned int value;

        rs_write( port , TrameInst , sizeof( TrameInst ));

        nb = readFrame(port , data);
```

```
        if(( nb == 25 ) || (nb == 26 ))
        {
                i = CalculCrc( data , 2 , data[2] );

                if( i != 0 )
                {
                        printf ( "Bad CRC\n" );
                        return;
                }

                if ( 0x13 == data[18] )
                {
                        bool Is2Octect = false;
                        if ( nb == 26 ) Is2Octect = true;

                        switch( data[12]  )
                        {
                        case 'B' :
                        case 'M' :
                                Is2Octect = false;
                                break;
                        case 'V' :
                                Is2Octect = true;
                                if( data[11] == 'A')  Is2Octect = false;
                                break;
                        }


                        if ( Is2Octect ) value = data[20]*256+ data[19];
                        else value = data[19];

                        printf(" value inst = %d \n", value );
                        return;
                }
        }
        printf(" no valid answer to read inst value\n" );
}


void read_stored_data ( void * port )
{
        int i,nb;
        unsigned char data[1024];
        unsigned int Moy[10];

        rs_write( port , Trame10M, sizeof( Trame10M ));

        nb = readFrame(port , data);

        if(( nb == 45 ) || (nb == 55 ))
        {
                i = CalculCrc( data , 2 , data[2] );

                if( i != 0 )
                {
                        printf ( "Bad CRC\n" );
                        return;
                }

                if ( 0x0D == data[18] )
                {
                        bool Is2Octect = false;
                        if ( nb == 55 ) Is2Octect = true;

                        switch( data[12]  )
                        {
                        case 'B' :
                        case 'M' :
                                Is2Octect = false;
                                break;
                        case 'V' :
                                Is2Octect = true;
                                if( data[11] == 'A')  Is2Octect = false;
                                break;
```

```
                        }

                        if( Is2Octect )
                        {
                                for ( i = 0 ; i < 10 ; i++ )
                                {
                                        Moy[i] = data[31+i*2] * 256 + data[30+i*2];
                                }
                        }
                        else
                        {
                                for ( i = 0 ; i < 10 ; i++ )
                                {
                                        Moy[i] = data[30+i];
                                }
                        }

                        for ( i = 0 ; i < 10 ; i++ )
                        {
                                printf(" value moy[%d]= %d \n", i , Moy[i] );
                        }
                        return;
                }
        }
        printf(" no valid answer to read stored values\n" );

}


// don't forget to change the define
#define SERIAL_PORT_NUMBER 79

int main(int argc, char* argv[])
{
        char Tc_Com[100];
        void * port = 0;

        sprintf(Tc_Com, "\\\\.\\COM%d" , SERIAL_PORT_NUMBER );
        port = rs_open ( Tc_Com);
        if ( port == 0 )
        {
                printf( "Serial Port COM%d not avaible\n" , SERIAL_PORT_NUMBER );
                return 0;
        }

        if( rs_setattr ( port ) == -1 )
        {
                printf( "Error config\n" , argv[1] );
                return 0;
        }

        while( 1 )
        {
                print_date();
                rs_flush( port);
                read_val_inst( port );
                rs_flush( port);
                read_stored_data( port );
        }
        rs_close( port );
        return 0;
}
```

10 CairSPM sensor

10.1 Last minute data of all parameter

For CAIRSPM sensor, first byte of REF must be set to **D** instead of **C**.
In addition to previous block length identifiers "B","M","V, a new identifier "P" is created to indicate "PACKET" data block only used in CAIRSPM.

- Query:

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x13
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	**0x44 =>CAIRSPM**
		0x44 => DUST
		0x50 => 'P' = PACKET
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x12 =>Last minute**
CRC	2 bytes	0x1C
		0xBF
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF			
STX	1 byte	0x02			
LG	**2 bytes**	**0x00,0x16**			
	7 bytes	0x2C			
		0x01			
		0x02			
		0x03			
		0x04			
		0x05			
		0x06			
REF	8 bytes	**0x44**			
		0x44			
		0x50 => 'P' = PACKET			
		0x01			
		0x00			
		0x00			
		0x00			
		0x04			
RSP	1 byte	**0x13**			
Last minute	22 bytes	0xE8,0x2B,0x12,0x05	PM2.5	µg/m3	float
		0x60,0x2C,0x56,0x01	PM10	µg/m3	float
		0x22,0x00	Temperature	1/10e °C	signed int16
		0x52	Humidity	0-100%	uchar
		0x15,0x04	Pressure	hPa	unsigned int16
		0x60	Battery charge	0-100%	uchar
		0x32	3WSolar efficiency	0-100%	uchar
		0x32	13WSolar efficiency	0-100%	uchar
		0x00,0x00	Spare_ANA1	millivolt	unsigned int16
		0x00,0x00	Spare_ANA2	millivolt	unsigned int16
		0x00,0x00	Spare_ANA3	millivolt	unsigned int16
END	2 byte	0x00			
		0xFF			
CRC	2 bytes	0x52			
		0x61			
ETX	1 byte	0x03			

Frame length = 22 + 25 = 47 bytes

This frame return all data from the card : DUST and technical.
The bytes are received from the most significant to least significant: int_32 = Rx [0] | Rx [1] << 8 | Rx [2] << 16 | Rx [3] << 24
32bit floating are coded "Little endian"
When the DUST module is not present, the measures PM2.5 and PM10 = NAN.

10.2 5 minutes archive DATA of all parameters

Command byte is **CMD** = **0x0C**

The parameter allowing the data download **PARAM** is built on a byte which value can vary and refer to several periods to download:

- 0x00 : 10 measurements (5 minutes averaged DATA), total of 50 minutes recovered DATA

Note : Others option are skipped from the list to avoid size of frames too important (0x01...0x07)

- Query for downloading 10 minutes data => **PARAM** = 0x00

SYNC	1 byte	0xFF
STX	1 byte	0x02
LG	1 byte	0x14
	7 bytes	0x30
		0x01
		0x02
		0x03
		0x04
		0x05
		0x06
REF	8 bytes	**0x44 => CAIRSPM**
		0x44 => DUST
		0x50 => 'P' = PACKET
		0xFF
		0xFF
		0xFF
		0xFF
		0xFF
CMD	1 byte	**0x0C => 5 min ARCHIVE DATA**
PARAM	1 byte	**0x00=> 10 values MAX downloadable**
CRC	2 bytes	0x63
		0xA8
ETX	1 byte	0x03

- Answer:

SYNC	1 byte	0xFF	Comment		
STX	1 byte	0x02			
LG	2 byte	**0x00** **0xDC**	**Max = 65535 data bytes**		
	7 bytes	0x2C			
		0x01			
		0x02			
		0x03			
		0x04			
		0x05			
		0x06			
REF	8 bytes	**0x44**	**CAIRSPM model**		
		0x44	**DUST**		
		0x50	**'P' = PACKET**		
		0x02			
		0x33			
		0x33			
		0x00			
		0x33			
RSP	1 byte	**0x0D**			
value n°1	22 bytes	0xE8,0x2B,0x12,0x05 0x60,0x2C,0x56,0x01 0x22,0x00 0x52 0x15,0x04 0x60 0x32 0x32 0x00,0x00 0x00,0x00 0x00,0x00	PM2.5 PM10 Temperature Humidity Pressure Battery charge 3WSolar efficiency 13WSolar efficiency Spare ANA1 Spare ANA2 Spare ANA3 **Averaged DATA, (Time-50min...Time-45)**	µg/m3 µg/m3 1/10°C 0-100% hPa 0-100% 0-100% 0-100% millivolt millivolt millivolt	float float signed int16 uchar unsigned int16 uchar uchar uchar unsigned int16 unsigned int16 unsigned int16
value n°2	22 bytes		**Averaged DATA, (Time-45min...Time-40)**		
value n°3	22 bytes		**Averaged DATA, (Time-40min...Time-35)**		
value n°4	22 bytes		**Averaged DATA, (Time-35min...Time-30)**		
value n°5	22 bytes		**Averaged DATA, (Time-30min...Time-25)**		
value n°6	22 bytes		**Averaged DATA, (Time-25min...Time-20)**		
value n°7	22 bytes		**Averaged DATA, (Time-20min...Time-15)**		
value n°8	22 bytes		**Averaged DATA, (Time-15min...Time-10)**		
value n°9	22 bytes		**Averaged DATA, (Time-10min...Time- 5)**		
value n°10	22 bytes		**Averaged DATA, (Time- 5min...Time - 0)**		

END	2 byte	0x00	
		0xFF	
CRC	2 bytes	0x69	
		0x0D	
ETX	1 byte	0x03	

PARAM = 0x00 => Frame length = 22 x 10 + 25 = 245 bytes.

When the DUST module is not present, the measures PM2.5 and PM10 = NAN.

10.3 Acquisition example for windows in c

```
#include <stdio.h>
#include <time.h>
#include <stdbool.h>

#include <windows.h>

#include "stdint.h"


#include <windows.h>

const int print_data = 1;

void print_trame(const unsigned char *Data, int nb)
{
	int i;
	for (i = 0; i < nb; i++)
	{
		printf(" %02X", Data[i]);
	}
}

void* rs_open(const char *name)
{
	HANDLE h;
	DCB dcb;

	h = CreateFileA(name,
		GENERIC_READ | GENERIC_WRITE,
		0, // must be opened with exclusive-access
		NULL, // default security attributes
		OPEN_EXISTING, // must use OPEN_EXISTING
		0, // not overlapped I/O
		NULL); // hTemplate must be NULL for comm devices

	if (INVALID_HANDLE_VALUE == h)
		return 0;

	ZeroMemory(&dcb, sizeof(dcb));
	dcb.DCBlength = sizeof(dcb);
```

```
	GetCommState(h, &dcb);

	dcb.fBinary = TRUE;

	dcb.BaudRate = CBR_9600;

	dcb.ByteSize = 8;
	dcb.Parity = NOPARITY;
	dcb.StopBits = ONESTOPBIT;
	//Setup the flow control
	dcb.fDsrSensitivity = FALSE;
	dcb.fOutxCtsFlow = FALSE;
	dcb.fOutxDsrFlow = FALSE;
	dcb.fOutX = FALSE;
	dcb.fInX = FALSE;

	if (!SetCommState(h, &dcb))
	{
		CloseHandle(h);
		return 0;
	}

	return (void*)h;
}

int rs_close(void* port)
{
	HANDLE h = (HANDLE)port;
	CloseHandle(h);
	return 0;
}


int rs_write(void* port, const void* data, int bytes)
{
	DWORD n = 0;
	HANDLE h = (HANDLE)port;

	if (!WriteFile(h, data, bytes, &n, NULL))
		return -1;

	if (print_data)
	{
		printf("\n=>");
		print_trame(data, bytes);
	}

	return n;
}

int rs_read(void* port, void* data, int bytes)
{
	DWORD n = 0;
	HANDLE h = (HANDLE)port;
```

```
	if (!ReadFile(h, data, bytes, &n, NULL))
		return -1;

	if (print_data)
	{
		print_trame(data, bytes);
	}

	return n;
}


int rs_flush(void* port)
{
	HANDLE h = (HANDLE)port;
	FlushFileBuffers(h);
	return 0;
}


int rs_bytesWaiting(void* port)
{
	//Check to see how many characters are unread
	COMSTAT stat;
	DWORD dwErrors = 0;
	HANDLE h = (HANDLE)port;

	if (!ClearCommError(h, &dwErrors, &stat))
	{
		return 0;
	}
	return stat.cbInQue;
}


unsigned char TrameInst[]= {0xFF, 0x02, 0x13, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05,
0x06, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x12, 0xAF, 0x88, 0x03};
unsigned char Trame10M[] = {0xFF, 0x02, 0x14, 0x30, 0x01, 0x02, 0x03, 0x04, 0x05,
0x06, 0x0FF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x0C, 0x00, 0x63, 0xA8,
0x03};


// RS232 define
#define	SYNC_BYTE		0xFF
#define	STX_BYTE		0x02
#define	ETX_BYTE		0x03
// reponse en mode simpe
#define	RSP_BYTE		0x30

#define COM_TIME_OUT 10
#define SIZE_RX 65536

int i_TimOut = 0;
int i_Eta = 0;
unsigned char Tc_RX[2048];
```

```
int i_Nmb_RX;

static int iP_RecComCairPol(unsigned char c_OctRec)
{
	int i;
	static unsigned char Octect_1 = 0;
	static unsigned char Octect_2 = 0;
	static unsigned char Octect_3 = 0;
	static unsigned char Octect_4 = 0;
	static unsigned char Octect_5 = 0;

	if (i_TimOut == 0)
	{
		i_Eta = 0;
		Octect_1 = 0;
		Octect_2 = 0;
		Octect_3 = 0;
		Octect_4 = 0;
		Octect_5 = 0;
	}

	i_TimOut = COM_TIME_OUT;

	if (i_Eta == 0)
	{ // attente de synchro
		Octect_1 = Octect_2;
		Octect_2 = Octect_3;
		Octect_3 = Octect_4;
		Octect_4 = Octect_5;
		Octect_5 = c_OctRec;

		if (Octect_5 == 0x2C)
		{
			if ((Octect_1 == 0xFF) && (Octect_2 == 0x02))
			{  // Octect_3 nombre de carateres du bloc data
				i = Octect_3 + Octect_4 * 256;
				if (i >= 19)
				{
					i_Nmb_RX = 5;
					Tc_RX[0] = Octect_1;
					Tc_RX[1] = Octect_2;
					Tc_RX[2] = Octect_3;
					Tc_RX[3] = Octect_4;
					Tc_RX[4] = Octect_5;

					i_Eta = 3 + i;

					Octect_1 = 0;
					Octect_2 = 0;
					Octect_3 = 0;
					Octect_4 = 0;
					Octect_5 = 0;
				}
			}
		}
```

```
		return 0;
	}

	// 1 : mémorise  l'octect
	Tc_RX[i_Nmb_RX++] = c_OctRec;

	if (i_Nmb_RX >= SIZE_RX)
	{ // erreur de reception
		i_Nmb_RX = 0;
		i_Eta = 0;
		Octect_1 = 0;
		Octect_2 = 0;
		Octect_3 = 0;
		Octect_4 = 0;
		Octect_5 = 0;
	}

	if (i_Nmb_RX >= i_Eta)
	{
		// initialisation
		i_Eta = 0;
		Octect_1 = 0;
		Octect_2 = 0;
		Octect_3 = 0;
		Octect_4 = 0;
		Octect_5 = 0;

		if (c_OctRec == ETX_BYTE)
		{
			return 1;
		}
	}
	return 0;
}



int readFrame(void * port, unsigned char * data, int lenbytes)
{
	int i,nb;
	unsigned char uc;


	if (print_data) printf("\n=>");

	i_TimOut = 0;
	i_Eta = 0;
	nb = 10;
	do
	{
		i = rs_bytesWaiting(port);
		if (i > 0)
		{
			rs_read(port, &uc, 1);
			if (iP_RecComCairPol(uc))
```

```
                        {
                                memcpy(data, Tc_RX, i_Nmb_RX);
                                return i_Nmb_RX;
                        }

                }
                else
                {
                        Sleep(200);
                        nb--;
                }
        } while (nb > 0);
        return 0;
}

unsigned int CalculCrc( unsigned char *data, unsigned int start , unsigned int
lg)
{
        unsigned int crc = 0;
        const unsigned int Poly = 0x8408;
        unsigned int  j, val, carry, i_bits;

        crc = 0;
        for( j = 0  ;  j < lg ; j++ )
        {
                val = data[j + start];
                crc = crc ^ val;
                for( i_bits = 0 ; i_bits < 8 ; i_bits++ )
                {
                        carry = crc & 1;
                        crc >>= 1;
                        if ( carry > 0 )   crc = crc ^ Poly;
                }

        }
        return crc;
}


unsigned char * GetTrame(unsigned char * Data, unsigned char *Reference)
{
        int i;
        if (Reference)
        {
                Data[10] = Reference[0];
                Data[11] = Reference[1];
                Data[12] = Reference[2];
                Data[13] = Reference[3];
                Data[14] = Reference[4];
                Data[15] = Reference[5];
                Data[16] = Reference[6];
                Data[17] = Reference[7];
        }
        else
        {
```

```
                Data[10] = 0xFF;
                Data[11] = 0xFF;
                Data[12] = 0xFF;
                Data[13] = 0xFF;
                Data[14] = 0xFF;
                Data[15] = 0xFF;
                Data[16] = 0xFF;
                Data[17] = 0xFF;
        }
        Data[Data[2]] = 0;;
        Data[Data[2] + 1] = 0;
        i = CalculCrc(Data, 2, Data[2] - 2);

        Data[Data[2]] = i & 0xFF;
        Data[Data[2] + 1] = ((i >> 8) & 0xFF);

        //i = CalculCrc(Data, 2, Data[2]); // check CRC if i == 0 ok

        return Data;
}

int32_t ToInt32(unsigned char * data, int pos)
{
        int32_t num;
        num = data[pos];
        num += data[pos + 1] << 8;
        num += data[pos + 2] << 16;
        num += data[pos + 3] << 24;

        return num;
}

float ToFloat(unsigned char * data, int pos)
{
        int32_t num;

        num = ToInt32(data, pos);

        return  *(float*)(&num);
}

int16_t ToInt16(unsigned char * data, int pos)
{
        int16_t  num;

        num = * (int16_t*)&data[pos];

        return num;
}

uint16_t ToUInt16(unsigned char * data, int pos)
{
        uint16_t  num;

        num = * (uint16_t*)&data[pos];
```

```
	return num;
}

void DecodeBloc(unsigned char * data)
{
	float f;
	int i;

	f = ToFloat( data, 0); printf("PM 2.5 = %g\n", f);
	f = ToFloat( data, 4);   printf("PM 10 = %g\n", f);
	i = ToInt16( data, 8);   printf("Temp (1/10 °C) = %d\n", i);
	i = data[10];            printf("Humidity 0-100% = %d\n", i);
	i = ToUInt16(data, 11);   printf("Pressure = %d\n", i);
	i = data[13];            printf("Battery charge 0-100% = %d\n", i);
	i = data[14];            printf("3W Solar charge 0-100% = %d\n", i);
	i = data[15];            printf("13W Solar charge  0-100% = %d\n", i);
	i = ToUInt16(data, 16);   printf("ANA1 = %d\n", i);
	i = ToUInt16(data, 18);   printf("ANA2 = %d\n", i);
	i = ToUInt16(data, 20);   printf("ANA3 = %d\n", i);
}

void read_stored_data(void * port, unsigned char *Reference)
{
	int i,nb;
	unsigned char data[1024];
	int pos;

	rs_write(port, GetTrame(Trame10M, Reference), sizeof(Trame10M));

	nb = readFrame(port , data , 2);

	if (nb <= 20)
	{
		printf(" \nno valid answer to read stored values\n");
		return;
	}

	i = CalculCrc( data , 2 , data[2] + data[3]*256 );

	if( i != 0 )
	{
			printf ( "\nBad CRC\n" );
			return;
	}

	if ( 0x0D != data[19] )
	{

			printf ( "\nBad RSP\n" );
			return;
	}

	i = 20;
	pos = 1;
```

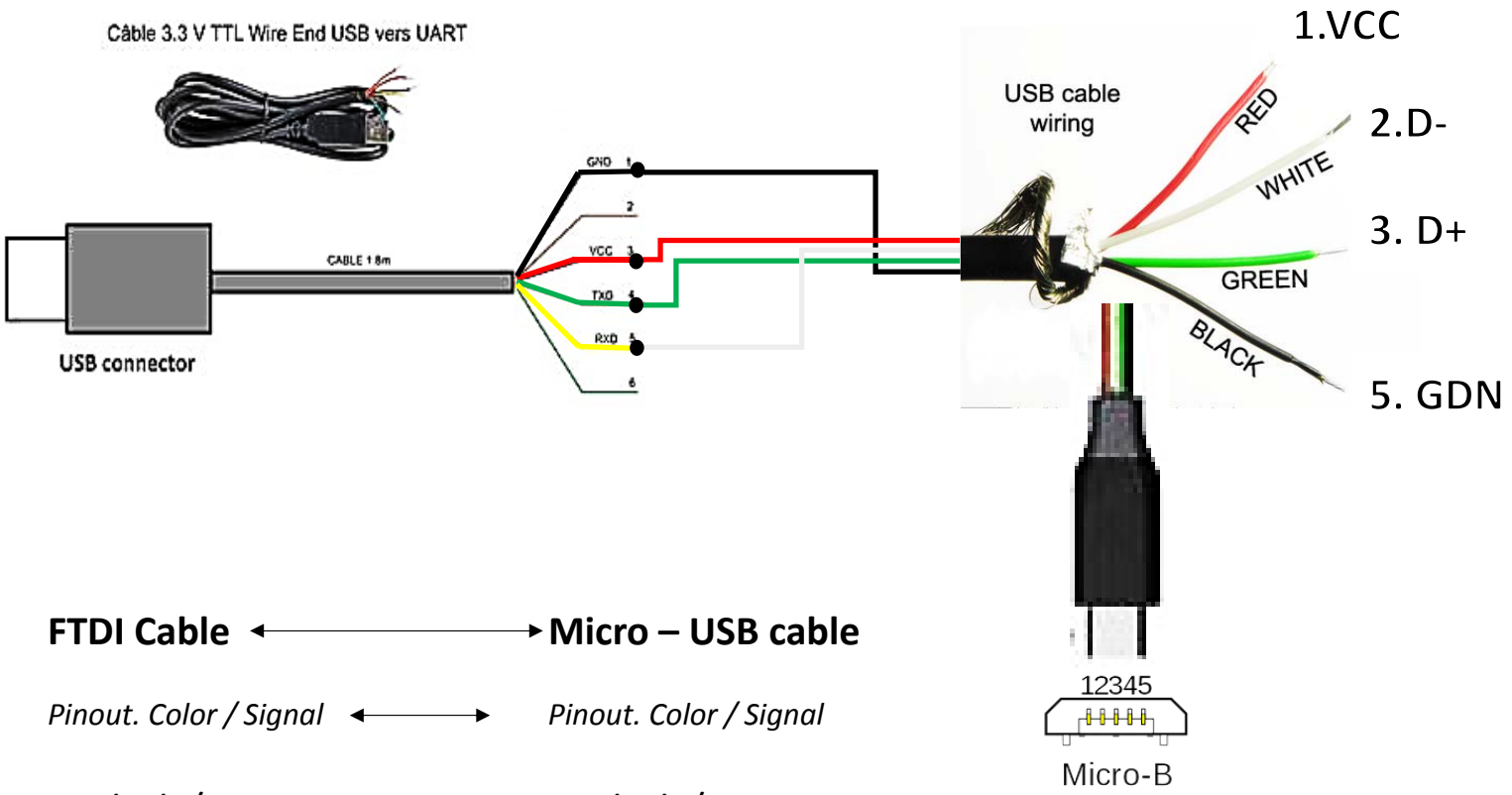
```
	do {

		printf("\n------ %d \n" , pos++);

		DecodeBloc(&data[i]);
		i += 22;
	} while ((i+22) < nb);

}


// don't forget to change the define
#define SERIAL_PORT_NUMBER 5

int main(int argc, char* argv[])
{
	char Tc_Com[100];
	void * port = 0;
	int j = 0;

	sprintf(Tc_Com, "\\\\.\\COM%d" , SERIAL_PORT_NUMBER );
	port = rs_open ( Tc_Com);
	if ( port == 0 )
	{
		printf( "Serial Port COM%d not avaible\n" , SERIAL_PORT_NUMBER );
		return 0;
	}

	while( 1 )
	{
		rs_flush(port);
		unsigned char REF1[]= { 'D', 'D', 'D', 0x01, 0x00, 0x00, 0x00, 0x01
};
		read_stored_data(port,REF1);

		Sleep(1000);
	}
	rs_close( port );
	return 0;
}
```

10.4 Decode data example

```
#include <stdio.h>
#include <time.h>
#include <stdbool.h>
#include <windows.h>
#include "stdint.h"
#include <windows.h>

int32_t ToInt32(unsigned char * data, int pos)
{
```

```
 int32_t num;
 num = data[pos];
 num += data[pos + 1] << 8;
 num += data[pos + 2] << 16;
 num += data[pos + 3] << 24;
 return num;
}

float ToFloat(unsigned char * data, int pos)
{
 int32_t num;
 num = ToInt32(data, pos);
 return  *(float*)(&num);
}

int16_t ToInt16(unsigned char * data, int pos)
{
 int16_t  num;
 num = * (int16_t*)&data[pos];
 return num;
}

uint16_t ToUInt16(unsigned char * data, int pos)
{
 uint16_t  num;
 num = * (uint16_t*)&data[pos];
 return num;
}

void DecodeBloc(unsigned char * data)
{
 float f;
 int i;

 f = ToFloat( data, 0);   printf("PM 2.5 = %g\n", f);
 f = ToFloat( data, 4);   printf("PM 10 = %g\n", f);
 i = ToInt16( data, 8);   printf("Temp (1/10 °C) = %d\n", i);
 i = data[10];            printf("Humidity 0-100% = %d\n", i);
 i = ToUInt16(data, 11);  printf("Pressure = %d\n", i);
 i = data[13];            printf("Battery charge 0-100% = %d\n", i);
 i = data[14];            printf("3W Solar charge 0-100% = %d\n", i);
 i = data[15];            printf("13W Solar charge  0-100% = %d\n", i);
 i = ToUInt16(data, 16);  printf("ANA1 = %d\n", i);
 i = ToUInt16(data, 18);  printf("ANA2 = %d\n", i);
 i = ToUInt16(data, 20);  printf("ANA3 = %d\n", i);
}


int main(int argc, char* argv[])
{
    unsigned char Tc[22] = { 0xF6, 0x98, 0x64, 0x42, 0xAE, 0x9A, 0x40, 0x43,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x53, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    DecodeBloc(Tc);
    return 0;
```

```
}

Output:
PM 2.5 = 57.1494
PM 10 = 192.604
Temp (1/10 °C) = 0
Humidity 0-100= 0
Pressure = 0
Battery charge 0-100= 83
3W Solar charge 0-100= 0
13W Solar charge 0-100= 0
ANA1 = 0
ANA2 = 0
ANA3 = 0
```



FTDI Cable		**Micro – USB cable**
Pinout. Color / Signal		*Pinout. Color / Signal*
1. Black / GND	⟷	5. Black / GND
3. Red / VCC	⟷	1. Red / VCC
4. Orange / TXD	⟷	3. Green / D+
5. Yellow / RXD	⟷	2. White / D-