NOTICE TECHNIQUE

CAIRSENS PM

MICROSENSOR

- JUILLET 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 des pages

CAIRSENS PM

1 GENERALITES – CARACTERISTIQUES

Le CAIRSENS PM est constitué d'un coffret de stockage (1), d'un capteur pour la mesure des PM1, PM2.5 et PM10 (2), d'un tube de prélèvement noir de longueur 8 cm (3).

(1) coffret de stockage, (2) capteur, (3) tube de prélèvement

Figure 1-1 – Présentation du CAIRSENS PM dans sa boite de stockage

Le capteur CAIRSENS PM peut être intégré dans une mini-station CAIRNET qui gère de façon autonome l'acquisition et la remontée de données (concentrations PM1, PM2.5, PM10, température et humidité relative interne au capteur) sur le CAIRCLOUD.

Le capteur CAIRSENS PM peut également être utilisé en version standalone. Il supporte deux protocoles de communication : UART et Modbus. Le CAIRSENS PM qui fonctionne en protocole UART délivre un signal de sortie UART TTL 3V.

ATTENTION : Par défaut, le CAIRSENS PM fonctionne en protocole UART (port micro USB B côté écran)

Pour une utilisation en version standalone, il convient d'utiliser les trames de requêtes à envoyer au CAIRSENS PM afin d'obtenir les données souhaitées :

- Pour le protocole UART :

  Se reporter au paragraphe « CAIRSPM Sensor » du document en annexe.

ATTENTION : – Le protocole UART décrit dans ce document ne donne accès qu'à deux mesures, les concentrations des PM10 et PM2.5. Les autres mesures, PM1, température et humidité relative interne au capteur, ne sont pas accessibles avec le protocole UART décrit dans ce document.

– NE PAS relier directement le CAIRSENS PM au port USB d'un PC. Prévoir un convertisseur UART vers USB (FTDI 3v3).

- Pour le protocole Modbus:

  Se reporter au document en annexe. Ce protocole permet d'accéder aux mesures des trois concentrations PM10, PM2.5 et PM1, ainsi qu'aux mesures de température et humidité relative interne au capteur.

Le CAIRSENS PM peut également fonctionner avec un outil logiciel spécifique de téléchargement, configuration, visualisation et exportation des données, le Cairsoft (nouvelle version 2020). Cet outil est disponible gratuitement dans la rubrique « Téléchargement » sur notre site web www.cairpol.com. Dans ce cas également, un câble convertisseur UART vers USB devra être utilisé pour relier le CAIRSENS PM au PC (voir plan en annexe).

1.2 GENERALITES

1.2.1 PRESENTATION

Le capteur CAIRSENS PM permet de mesurer en continu et en temps réel les concentrations en particules fines PM1, PM2.5 et PM10 (en µg/m3) présentes dans l'air ambiant. Il effectue les mesures de ces concentrations à titre indicatif : l'incertitude totale inférieure à 50% est en accord avec les objectifs de qualité de données fixés par la Directive européenne 2008/50/CE concernant la qualité de l'air ambiant et un air pur pour l'Europe.

Le capteur CAIRSENS PM est calibré en laboratoire de qualification métrologique par ENVEA avec une durée de validité d'un an. Il est compact et consomme très peu d'énergie. Il fournit des concentrations toutes les minutes.

Le capteur CAIRSENS PM peut être intégré dans des mini stations CAIRNET pour le suivi de plusieurs polluants, ou encore s'intégrer de façon personnalisée dans un réseau de supervision de la qualité de l'air.

Les applications du capteur CAIRSENS PM les plus usuelles sont les suivantes :

- Surveillance de la qualité de l'air intérieur et extérieur : villes intelligentes, bords de route et tunnels, écoles, aéroports, terminaux de navires...
- Fourniture de données pour modéliser la dispersion atmosphérique.
- Santé et sécurité : mines, sites industriels, construction.
- Prévision des émissions en limite séparative de sites industriels.

1.2.2 DESCRIPTION

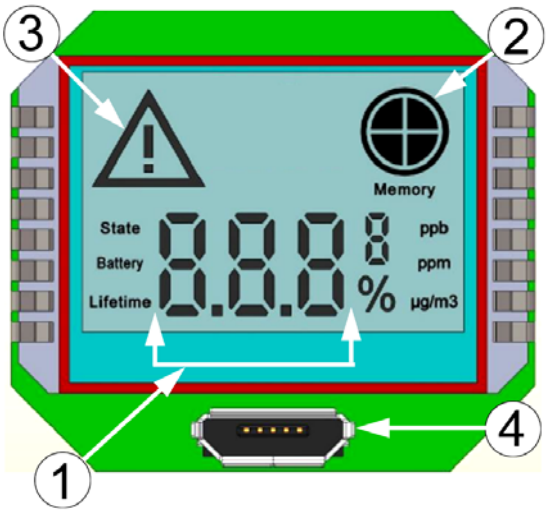
1.2.2.1 Face avant

Voir Figure 1-2.

La face avant, protégée par une plaque de polycarbonate, est équipée de :

– L'écran LCD qui affiche :
  - La concentration mesurée (1), représentée par 3 chiffres et 1 exposant (voir Chapitre 2.).
  - L'état de la capacité mémoire utilisée (2), par fraction de 25% : lorsque toute la mémoire est complètement utilisée, les quarts de cercle sont pleins.
  - Les éventuels défauts de fonctionnement du CAIRSENS PM (3).

– Le port micro-USB B (4), protégé par un capuchon amovible, qui permet :
  - L'alimentation du capteur en 5 VDC.
  - La communication en UART CAIRPOL et UART MODBUS (esclave).



(1) concentration mesurée, (2) mémoire utilisée, (3) erreurs de fonctionnement, (4) port micro-USB B

Figure 1-2 – Face avant du CAIRSENS PM

1.2.2.2 Corps du capteur

Le corps du capteur contient les éléments nécessaires à la mesure. Il est protégé par un boitier externe gris en aluminium anodisé EN AW-6060-T6 qui assure le maintien en place des éléments.

Ce capteur CAIRSENS PM est exempté de batterie Lithium interne, seule une pile bouton assure le maintien de l'horodatage.

Le circuit fluide est d'une grande simplicité : l'air à analyser est aspiré à travers l'entrée échantillon par un ventilateur interne et est rejeté à travers l'orifice situé en face arrière.

ATTENTION : NE PAS obstruer l'entrée du gaz échantillon, ni sa sortie par l'orifice sur la face arrière.

Figure 1-3 – Orifice de sortie d'air (1)

Figure 1-4 – Vue de l'entrée échantillon (2)

NOTE : Connecter le tube de prélèvement sur le raccord visible (2) sous l'ouverture du corps du CAIRSENS PM.

Les informations suivantes figurent sur l'étiquette du CAIRSENS PM : le polluant mesuré (1), la gamme de mesure (2), la tension d'alimentation pour le raccordement (3), la référence du CAIRSENS PM (4) et son numéro de série (5).

(1) polluant mesuré, (2) gamme de mesure, (3) tension d'alimentation,
(4) référence A40-0414 du CAIRSENS PM, (5) n° de série

Figure 1-5 – Etiquette du CAIRSENS PM

Le capteur ne doit pas être éliminé dans une poubelle traditionnelle, mais dans une structure de récupération et de recyclage appropriée.

1.2.3 MODES DE FONCTIONNEMENT

1.2.3.1 En standard

- Via l'utilisation de la communication UART ou MODBUS : intégré dans une solution personnalisée avec centralisation des données sur un DAS à des fins de supervision et de suivi de la qualité de l'air en l'associant avec d'autres types de mesures (météo, bruit, …).
- Intégration dans une mini-station CAIRNET 3.0.

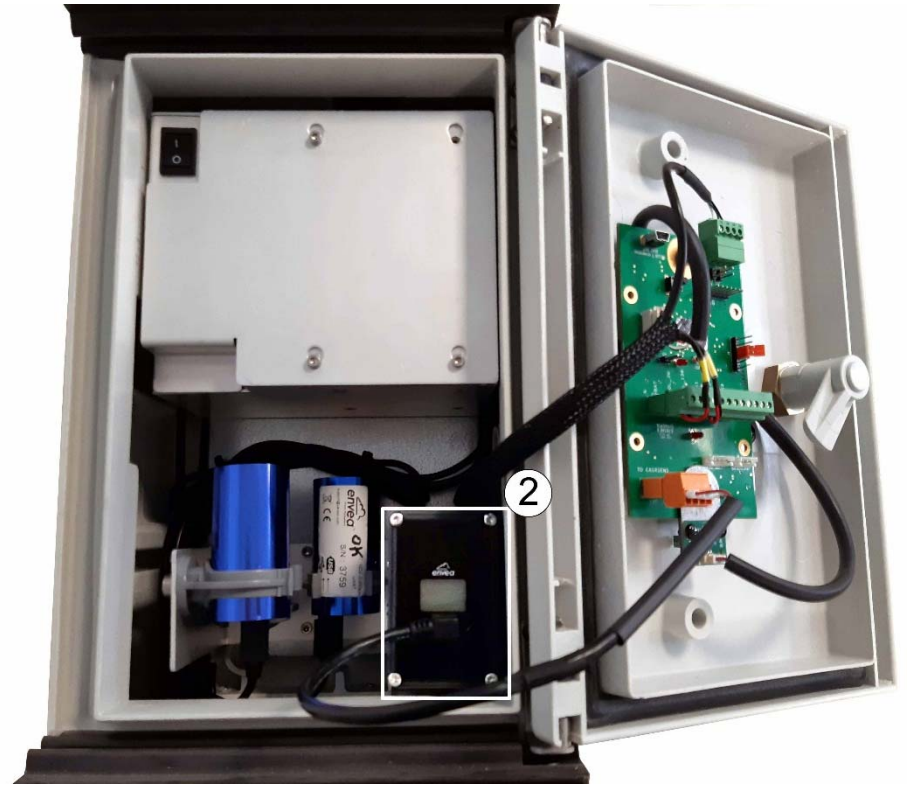
  La mini-station CAIRNET 3.0 peut être alimentée de deux manières différentes :

  - Soit avec une alimentation compatible 8-30 V,
  - Soit de manière autonome en incluant l'option d'alimentation par batterie rechargeable sur panneaux photovoltaïque et un module de communication sans fil cellulaire 3G/4G (Radio ou GPRS), permettant une mesure de remontée de données sur le CAIRCLOUD.

(1) CAIRSENS PM

Figure 1-6 – Intégration dans la mini-station CAIRNET 3.0

NOTE : Le CAIRSENS PM est compatible avec les anciens produits CAIRNET 2.0



(2) CAIRSENS PM

Figure 1-7 – Intégration dans une mini station CAIRNET 2.0

Remarque sur la longueur du tube de prélèvement :

- Pour une intégration dans un CAIRNET V3.0 : le tube fournit avec le capteur doit être coupé pour avoir une longueur de 2,5 cm
- Pour une intégration dans un CAIRNET V2.0 : le tube fournit avec le capteur doit être coupé pour avoir une longueur de 4 cm

1.2.3.2 En option

Le kit SAV-K-000290 contient toutes les pièces nécessaires pour l'intégration du CAIRSENS PM dans un CAIRNET V2.0.

1.2.4 EQUIPEMENT ASSOCIE

Le CAIRSENS PM peut être associé aux équipements suivants (non fournis) :

- Système d'acquisition et de gestion des données.

1.3 CARACTERISTIQUES

1.3.1 CARACTERISTIQUES TECHNIQUES

Les caractéristiques techniques du CAIRSENS PM sont :

Matières particulaires mesurées	PM1, PM2.5 & PM10
Gamme de mesure (3)	0 – 1 000 µg/m3
Diamètres (ø) des particules détectées	0.3 – 10 µm
Limite de détection certifiée* (2)	< 5 µg/m3
Résolution d'affichage	0.01 µg/m3
Linéarité (2)	$R^2 > 0.75$
Incertitude entre les capteurs	< 5 µg/m3
Précision (pente) (2)	0.7 à 1.3
Conditionnement de l'échantillon	Débit d'air contrôlé, Air chauffé au-delà de 60% d'humidité relative.
Effet de la température	< 0.01 µg/m3/°C
Technologie	Diffusion de la lumière laser
Température de fonctionnement	-20 to 70 °C
Humidité relative de fonctionnement	0 à 95 HR % (sans condensation)
Pression atmosphérique de fonctionnement	500 à 1 500 mbar

(2) Selon l'évaluation de notre laboratoire : mesure des moyennes journalières des PM2.5 par comparaison avec une référence

(3) Equivalent de sable de l'Arizona

Les spécifications pour l'intégration du CAIRSENS PM dans les systèmes clients sont :

Durée de vie du capteur CAIRSENS-PM	1 an en fonctionnement continu
Alimentation électrique nominale	– 5V DC / 500 mA, – Port USB d'un PC ou d'un chargeur externe (non fourni).
Consommation électrique	250 mA max pour 5 VDC
Méthode de prélèvement du gaz	– Circulation d'air régulée avec un ventilateur, – Débit de 2.5 L/min
Communications et identifiants de connexion Entrée/Sortie	UART Cairpol et UART Modbus (esclave) via le port micro-USB B
Affichage LCD	– Concentration en µg/m3, – Durée de vie résiduelle du capteur, – Etats de fonctionnement, – Mémoire disponible
Carte de commande et de traitement de données	Microprocesseur interne pour l'acquisition et le traitement de données, chronomètre intégré
Capacité de stockage des données (interne)	– 2 jours de données 1 minute, – 30 jours de données 15 minutes – ou 120 jours de données 60 minutes
Mode de téléchargement des données	– Intégration personnalisée / DAHS – Mini-station CAIRNET (export de données sur le CairCloud®) (option)

Conformité du CAIRSENS PM à la règlementation environnementale:

Sécurité électrique	NF EN 61010-1: 2010
Compatibilité électromagnétique	NF EN 61326-1: 2013
Indice de protection	IP 42 (selon la norme CEI 60529)

1.3.2 CARACTERISTIQUES DE STOCKAGE

Les conditions de stockage du CAIRSENS PM à respecter sont :

Température (°C)	-20 à 70
Humidité relative (% HR)	0 à 95 (sans condensation)
Pression (mbar)	500 à 1500

1.3.3 CARACTERISTIQUES D'INSTALLATION

1.3.3.1 Liaisons entre appareils

Le capteur CAIRSENS-PM met en œuvre les liaisons externes et les alimentations comme illustré Figure 1-8 :

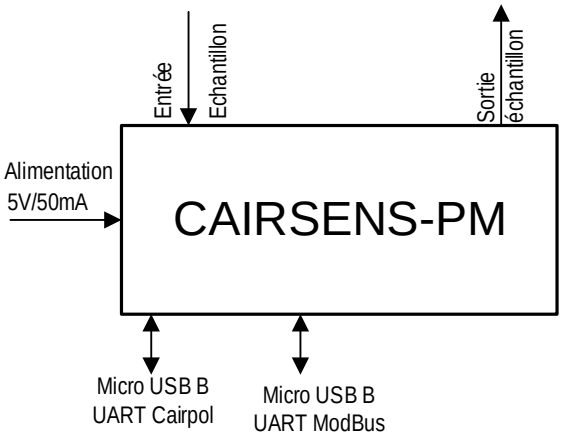


Figure 1-8 – Liaisons entre appareils pour CAIRSENS-PM en fonctionnement standalone

1.3.3.2 Encombrement et masse

L'appareil se présente sous la forme d'un pavé :

Longueur : 90 mm

Largeur : 45 mm

Hauteur : 70 mm

Masse : 370 grammes

1.3.3.3 Manutention et stockage

Le capteur CAIRSENS-PM doit être manipulé avec précaution.

Il doit être conservé dans le coffret prévu à cet effet.

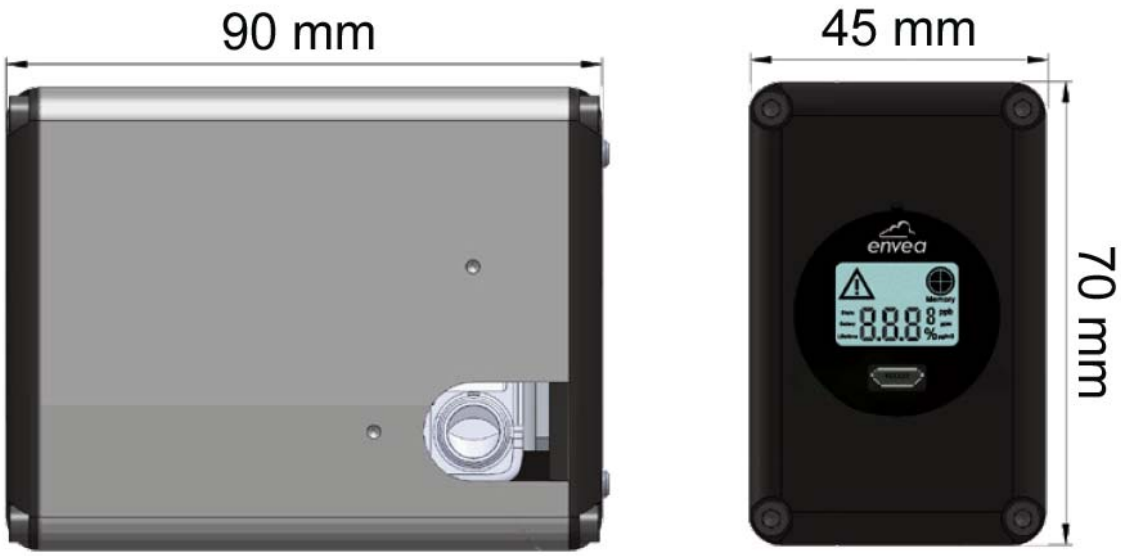


Figure 1-9 – Cotes d'encombrement du CAIRSENS PM

2 PRINCIPE DE FONCTIONNEMENT ET MESURE

2.1 PRINCIPE DE MESURE

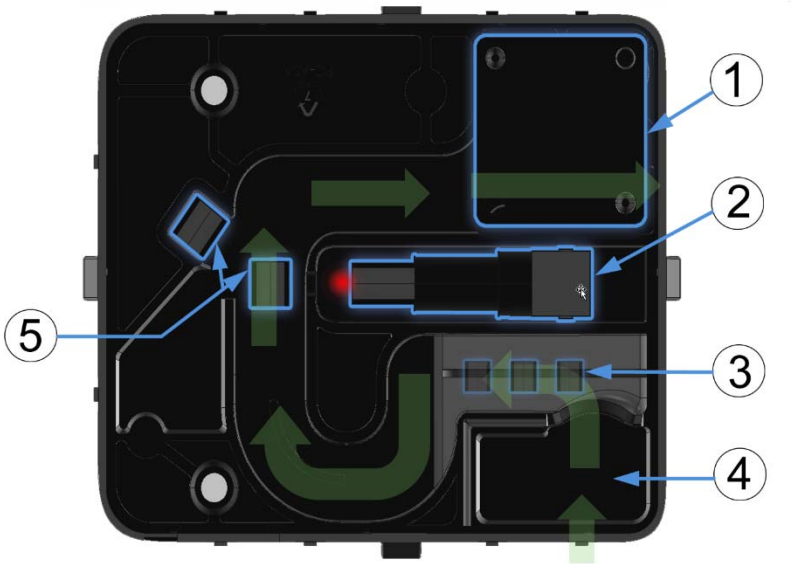
Le CAIRSENS PM effectue des mesures grâce à la présence d'un capteur de particules capable de mesurer simultanément les concentrations PM10, PM2.5 et PM1 (µg/m3).

Les particules prélevées dans la chambre de mesure traversent un faisceau laser continu (2) et émettent de la lumière dans toutes les directions suivant le principe de la diffusion lumineuse. Cette lumière diffusée est analysée pour calculer une concentration massique en µg/m3 pour chaque fraction granulométrique : PM1, PM2.5 et PM10.

Le prélèvement d'air est effectué à l'aide d'un micro-ventilateur (1) contrôlé pour maintenir le débit à 2,5 L/min.

Un système de chauffage de l'échantillon permet de maintenir l'humidité relative mesurée dans la chambre de mesure sous le seuil de 60%, au-delà duquel les caractéristiques optiques des particules sont altérées et la mesure du capteur est faussée.

Ce système de chauffage permet ainsi de s'affranchir des effets indésirables d'un fort taux d'humidité relative sur la mesure des PM10, PM2.5 et PM1.



(1) Micro ventilateur géré hautement fiable, (2) Laser à haute performance, (3) Gestion de l'humidité relative et de la température, (4) entrée d'air, (5) détecteurs optiques.

Figure 2-1 – Capteur de mesure du CAIRSENS PM

Les courbes ci-dessous donnent des exemples de résultats obtenus avec le capteur :

- La courbe ci-dessous montre l'évolution en fonction du temps des concentrations de PM2.5 (moyennes journalières) mesurées simultanément sur un même site de mesure par un CAIRSENS PM et une jauge beta MP101M (AMS certifiée US EPA et QAL1 EN16450).

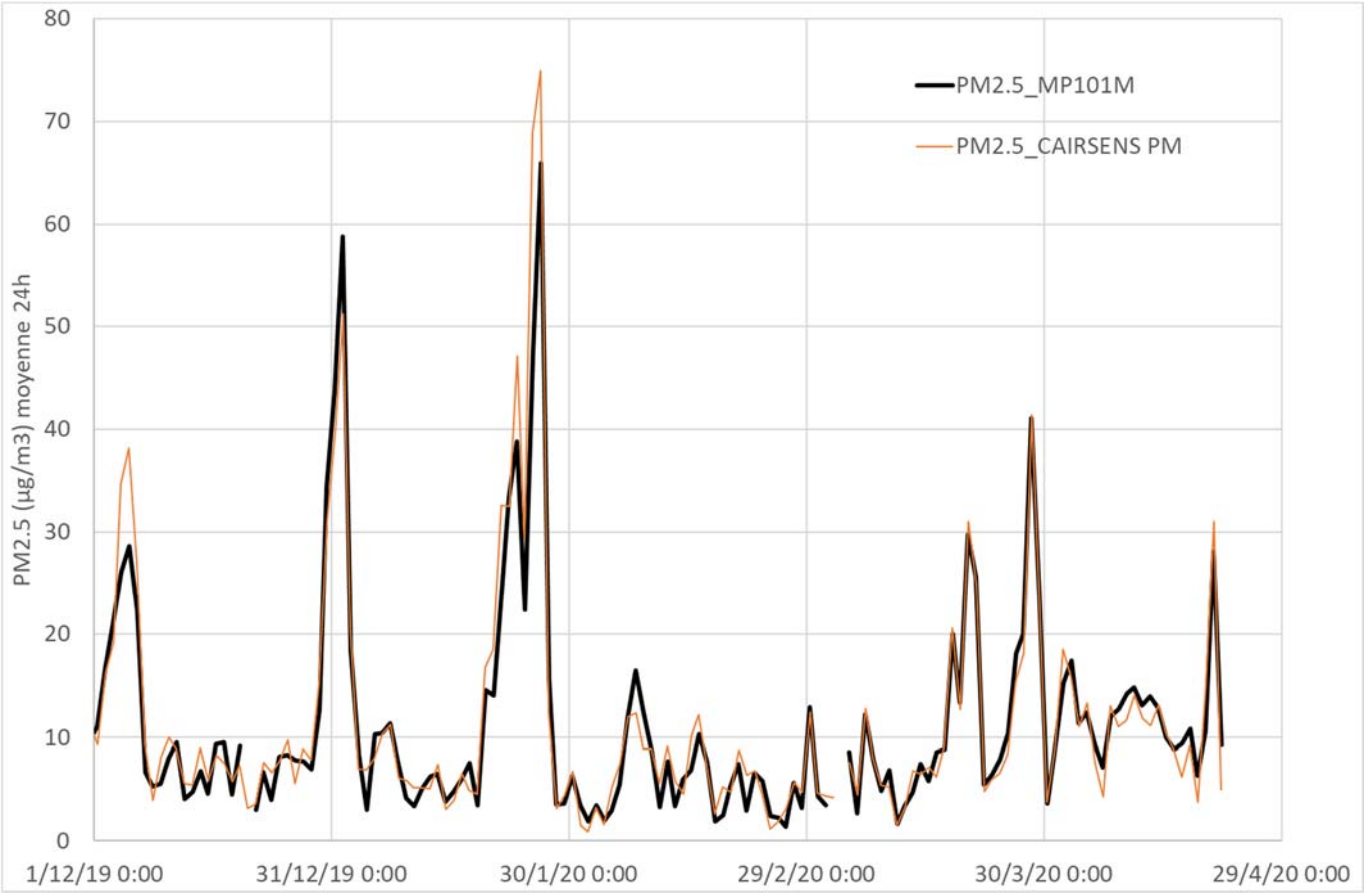


Figure 2-2 – Evolution des concentrations PM 2.5

– La courbe ci-dessous permet de comparer des concentrations PM2.5 (moyennes journalières) mesurées simultanément sur un même site de mesure par un CAIRSENS PM et une jauge beta MP101M (AMS certifiée US EPA et QAL1 EN16450) de juillet 2019 à avril 2020 (262 jours de mesure).

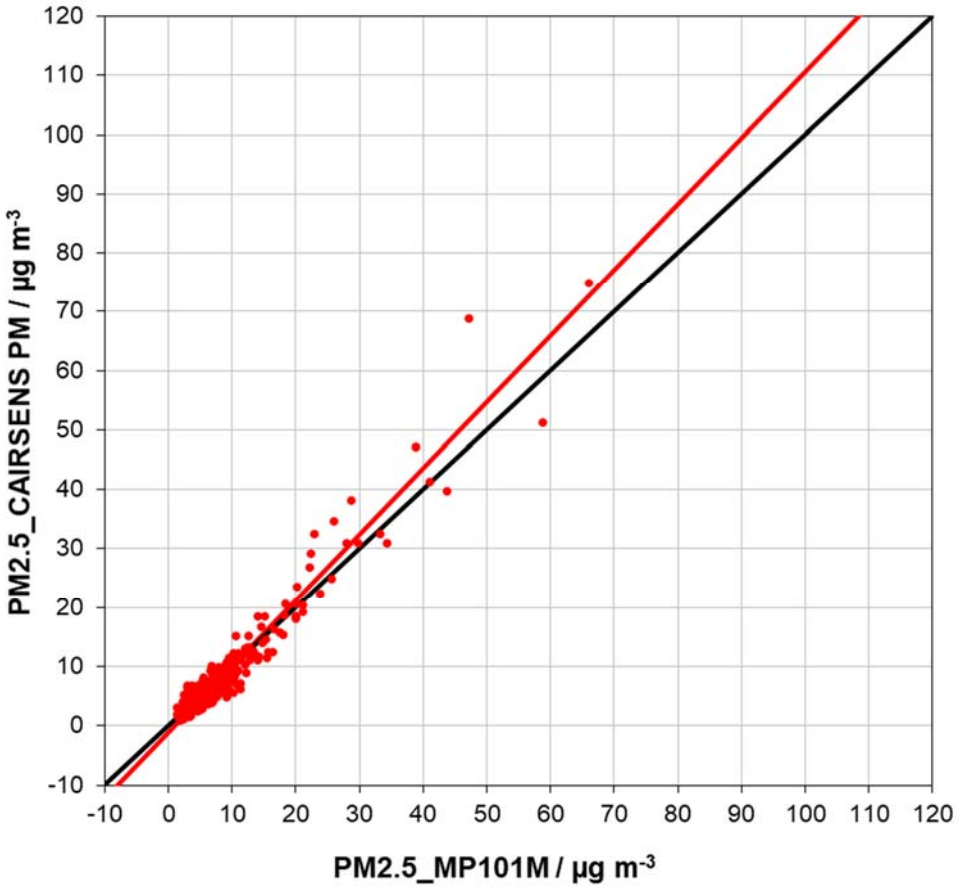


Figure 2-3 – Comparaison des concentrations PM2.5

NOTE : L'incertitude élargie ainsi obtenue est égale à 21.9 % (l'objectif pour la mesure indicative étant de 50%).

La pente et l'offset de la régression linéaire ainsi obtenue sont respectivement égaux à 1.1 et -1.1. Ces valeurs sont très satisfaisantes pour une mesure indicative.

2.2 MESURE

Les mesures réalisées par le CAIRSENS PM présentent les caractéristiques suivantes :

- Moyenne glissante des mesures du capteur sur 60 secondes. Cette moyenne est recalculée toutes les 10 secondes.
- Défilement successif des mesures PM10, PM2.5, PM1 à l'écran.
- Fréquence des mesures enregistrées dans la mémoire du CAIRSENS PM paramétrable.

Lecture de la concentration sur l'afficheur :

Voir **Figure 2-4**.

La valeur numérique de la concentration mesurée est donnée par la formule :

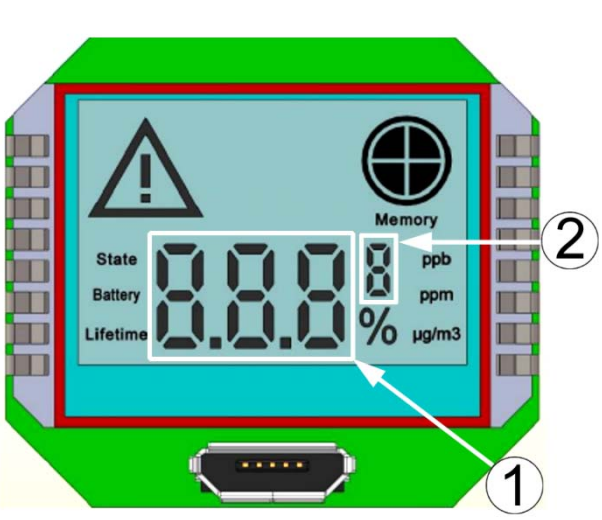
$$XXX \times 10^{y}$$

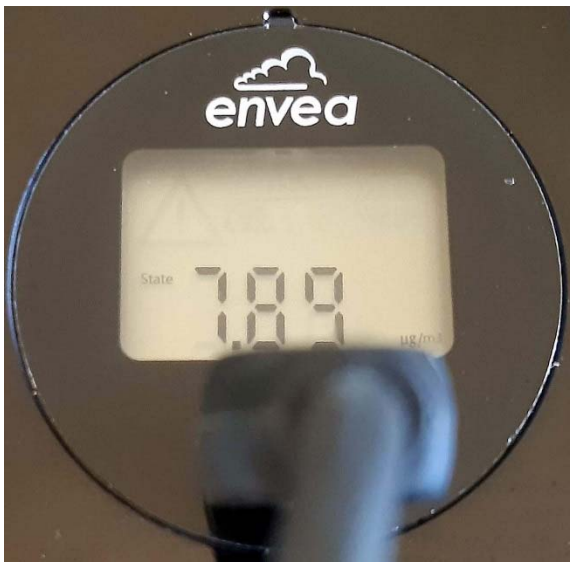
Où :

- XXX est la valeur de mesure affichée sur les 3 chiffres de l'écran (1)
- ✖ est l'opérateur de multiplication
- 10^{y} est le facteur multiplicatif à appliquer à XXX. Sur l'écran, la valeur de « y » est affichée en (2).

A titre d'exemple, le tableau ci-dessous donne la valeur de la concentration pour des valeurs de y de 0 à 2 :

Valeur de y	Valeur de 10^{y}	Valeur de la concentration lue
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) affichage de la valeur mesurée sur 3 chiffres XXX, (2) valeur de « y » dans la formule : $XXX \times 10^{y}$

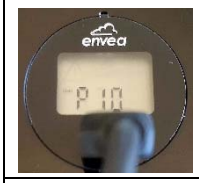
Figure 2-4 – Affichage de la concentration (en µg/m3)

Comme vu précédemment, le capteur mesure simultanément les concentrations PM10, PM2.5 et PM1. L'affichage des valeurs défile en série de séquence identique sur l'écran, dans l'ordre indiqué ci-dessous :

P10 ⇨ valeur des PM10 uniquement ⇨ P2.5 ⇨ valeur des PM2.5 uniquement ⇨ P1 ⇨ valeur des PM1 uniquement.

Lorsque la séquence est terminée, une nouvelle séquence commençant par P10 défile.

Les mesures sont exprimées en en µg/m3.

⇨ Séquence d'affichage des mesures ⇨					
					
Pour PM10	[C] PM10 = 7,89 µg/m3	Pour PM2.5	[C] PM2.5 = 1,45 µg/m3	Pour PM1	[C] PM1 = 1,36 µg/m3

3 FONCTIONNEMENT

3.1 MISE EN SERVICE

3.1.1 PREMIERE MISE EN FONCTIONNEMENT

La mise en fonctionnement du CAIRSENS PM se fait dès son alimentation électrique : le CAIRSENS PM commence à mesurer dès sa mise en fonctionnement. Les mesures sont immédiatement affichées à l'écran et mémorisées automatiquement. La sauvegarde des mesures est réalisée en permanence dans la mémoire interne du CAIRSENS PM.

Par défaut, le CAIRSENS PM est en mode de mesure continue, et la période de mesure (ou pas de temps) est de 1 minute. Elle est modifiable à l'aide de commandes spécifiques des protocoles UART et Modbus (voir les documentations en annexe) ou par le logiciel CAIRSOFT.

Lorsque la mémoire du CAIRSENS PM est pleine, le capteur continue à fonctionner normalement, mais il enregistre les nouvelles mesures en écrasant les mesures les plus anciennes.

Au démarrage, tous les pictogrammes s'affichent par défaut (Figure 3-1) puis les messages « init » et « notr » (NOT READY) apparaissent successivement à l'écran:

Figure 3-1 – Ecrans d'affichage au démarrage du CAIRSENS PM

Remarque : Il est possible que des alarmes « fan » ou « trh » s'affichent au démarrage pendant quelques secondes, puis disparaissent automatiquement.

L'étalonnage et la mesure du CAIRSENS PM sont garantis valables durant 12 mois après sa livraison.

La durée de vie du capteur est de 12 mois (8760 h d'utilisation). Lorsque cette durée est expirée, l'affichage sur l'écran est : cAL (Figure 3-2). Il est alors nécessaire de procéder au renouvellement du capteur.

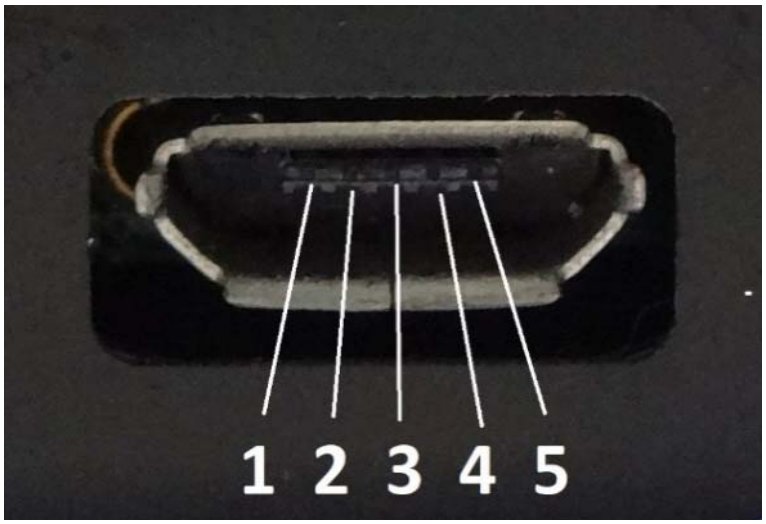
Figure 3-2 – Ecran d'affichage en fin de durée de vie

3.1.2 HEURE, DATE ET HORODATAGE DES DONNEES MESUREES

L'horloge interne du CAIRSENS PM est préréglée en usine, une pile bouton assure la sauvegarde de l'horloge.

3.1.3 CABLAGE DU CAIRSENS PM POUR INTEGRATION

Les indications concernant le câblage nécessaire pour l'intégration du CAIRSENS PM dans un système sont données ci-dessous :



Broche Micro USB B	Description	Couleur du câble
1	VDC +5V	Rouge
2	Data -	Blanc
3	Data +	Vert
4	Non utilisé	Marron
5	GND Ground	Noir

Figure 3-3 – Indications de câblage pour l'intégration en système

3.1.4 PROTOCOLES DE COMMUNICATIONS

Le capteur CAIRSENS PM supporte deux protocoles de communication UART-CAIRPOL (par défaut) et UART-Modbus.

ATTENTION : Par défaut le CAIRSENS PM fonctionne en protocole UART du côté écran (port micro USB B).

Pour modifier le protocole de communication sur le port micro-USB du CAIRSENS PM :

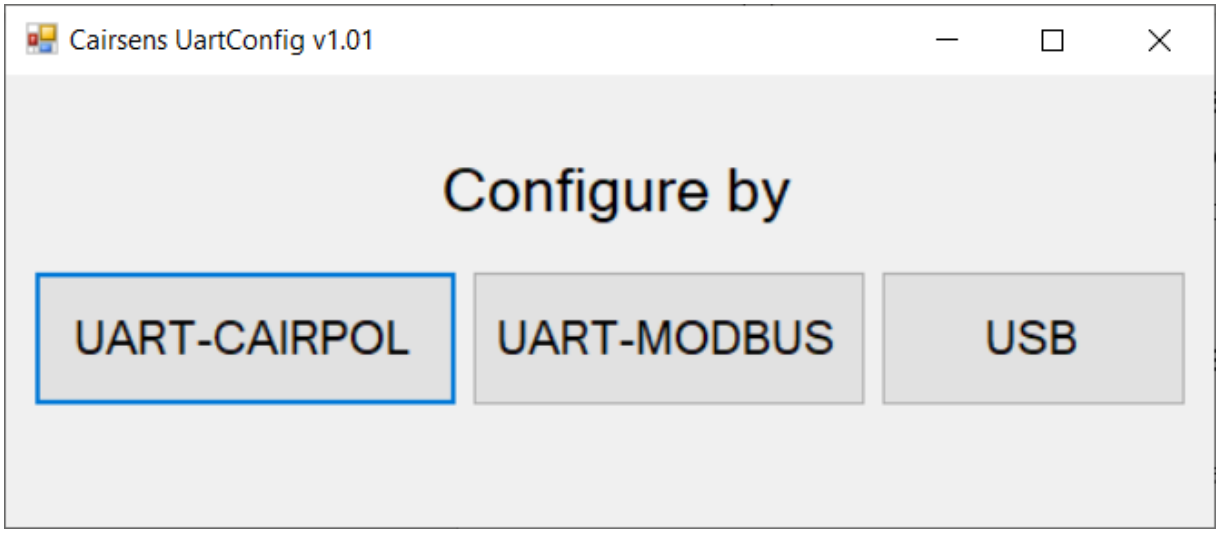
1. Connecter le CAIRSENS PM au PC en utilisant un convertisseur UART vers USB (FTDI 3v3) : voir plan en annexes.

Utiliser de préférence un poste fixe plutôt qu'un PC portable pour avoir une tension d'alimentation suffisante pour cette mise à jour du CAIRSENS PM.

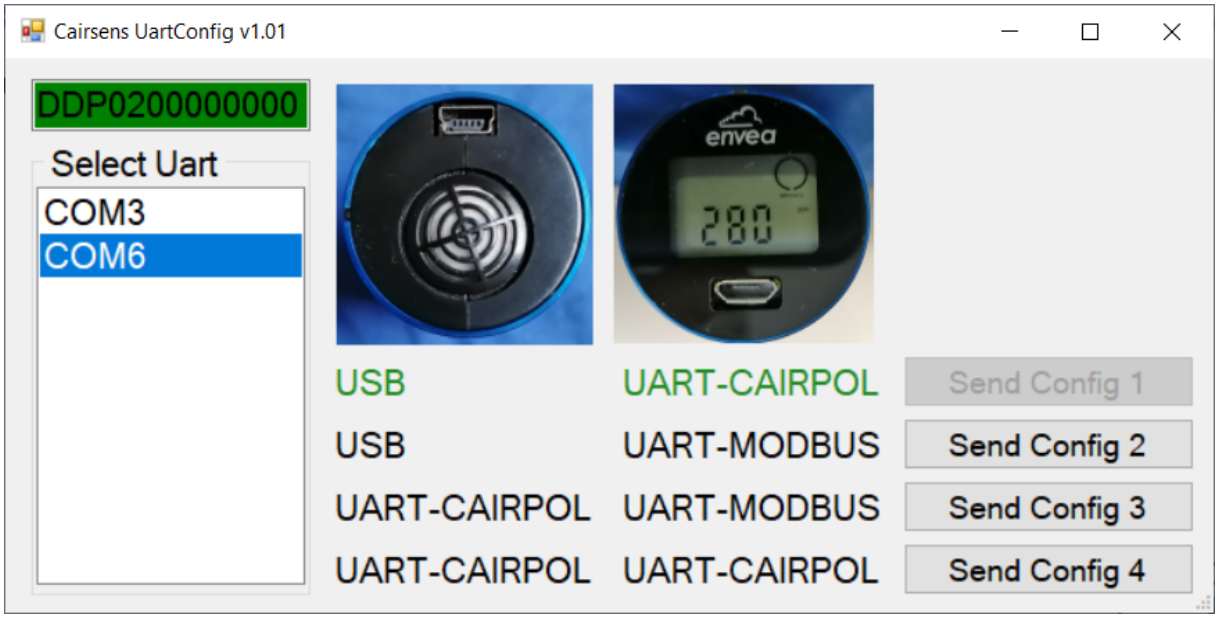
2. Exécuter l'application **uartconfig.exe** téléchargeable gratuitement sur le site web www.cairpol.com, au même endroit que le CAIRSOFT : rubrique « Téléchargement », onglet « Documentation technique ».

3. Dans la fenêtre qui s'ouvre au lancement de l'application, sélectionner « UART-CAIRPOL » ou « UART-MODBUS » en fonction de la configuration de départ du CAIRSENS PM.

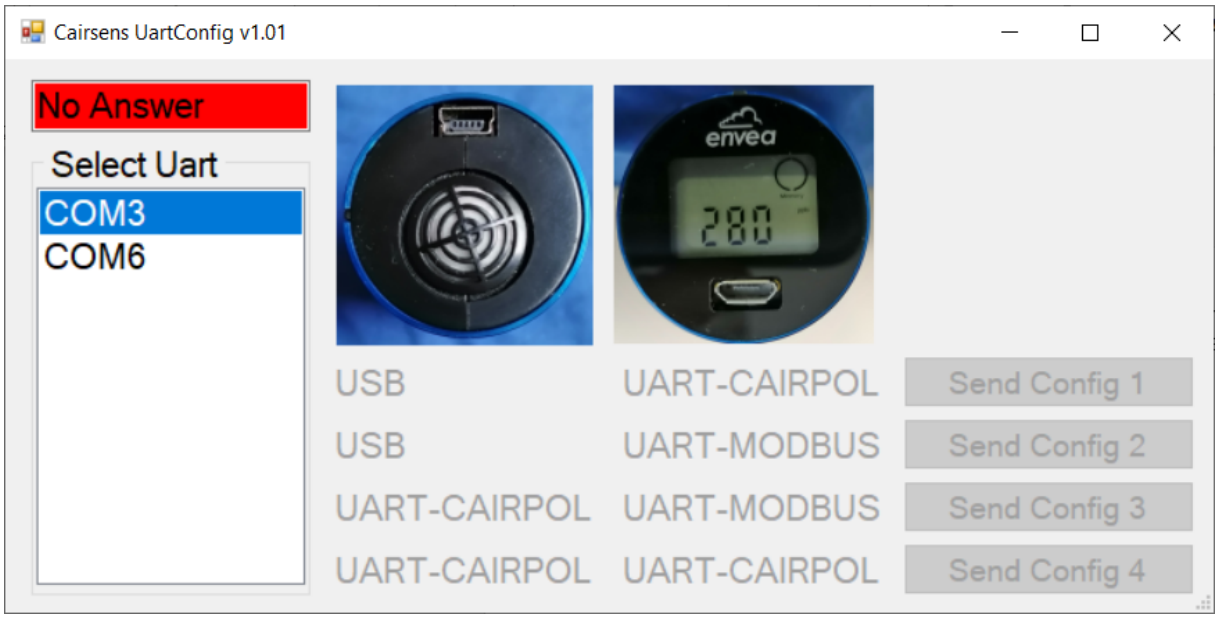
Par défaut le CAIRSENS PM sort d'usine configuré avec le protocole UART CAIRPOL, dans ce cas il faut donc cliquer sur « UART-CAIRPOL » :



4. Dans la nouvelle fenêtre, sélectionner le port de COM sur lequel se trouve connecté le CAIRSENS PM. Dans l'exemple ci-dessous, il faut cliquer sur « COM6 ». Une fois le capteur reconnu, son numéro de série (de la forme DDP02XXXXXXXX) apparait dans la cellule en haut à gauche de la fenêtre et la cellule devient verte.

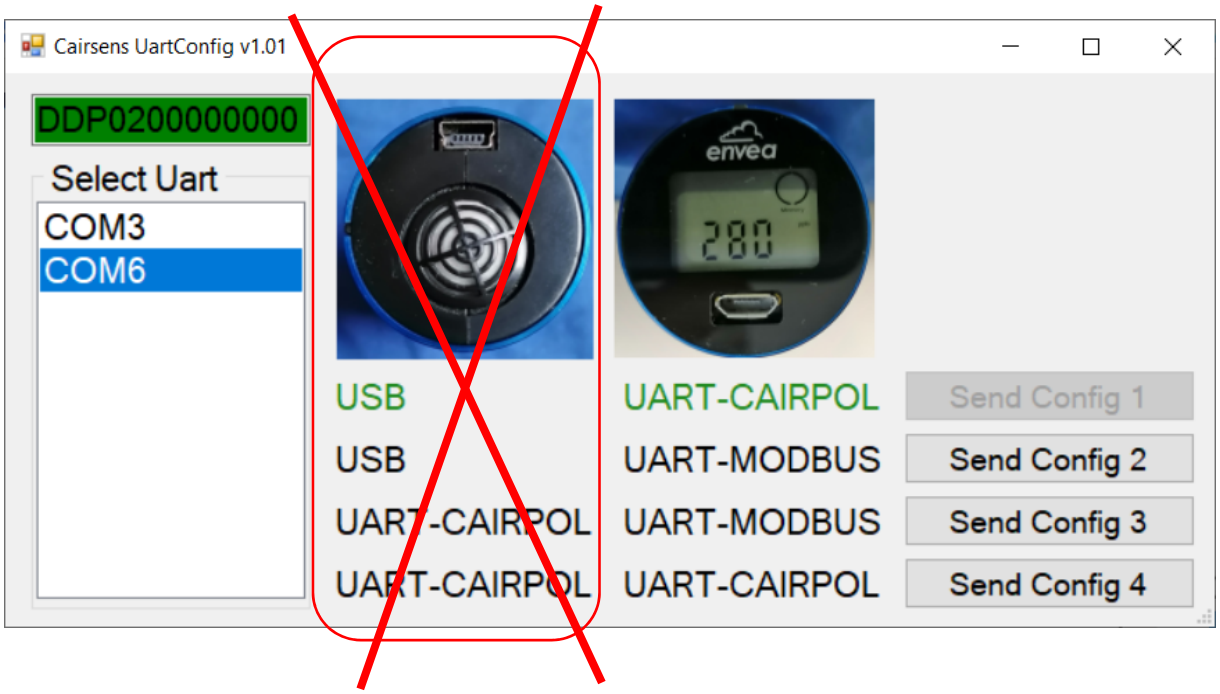


Si le mauvais port de COM est sélectionné ou si le capteur n'est pas détecté, la cellule en haut à gauche de la fenêtre indique « No Answer » et devient rouge :



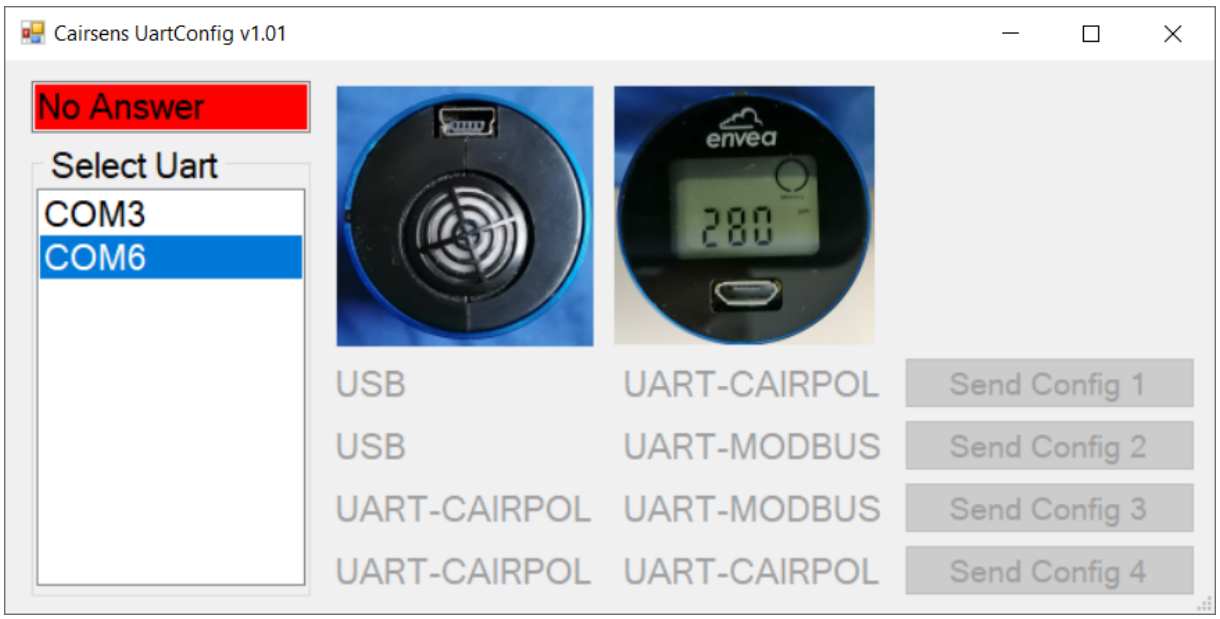
5. Lorsque le capteur est bien détecté, sa configuration actuelle apparait en vert sur la partie droite de l'écran (ici : « UART-CAIRPOL » dans la deuxième colonne).

ATTENTION : Pour le CAIRSENS PM, il ne faut considérer que la deuxième colonne qui est placée sous la photo de l'écran. La première colonne concerne uniquement le port mini-USB situé à l'arrière des capteurs de gaz de la série CAIRSENS.



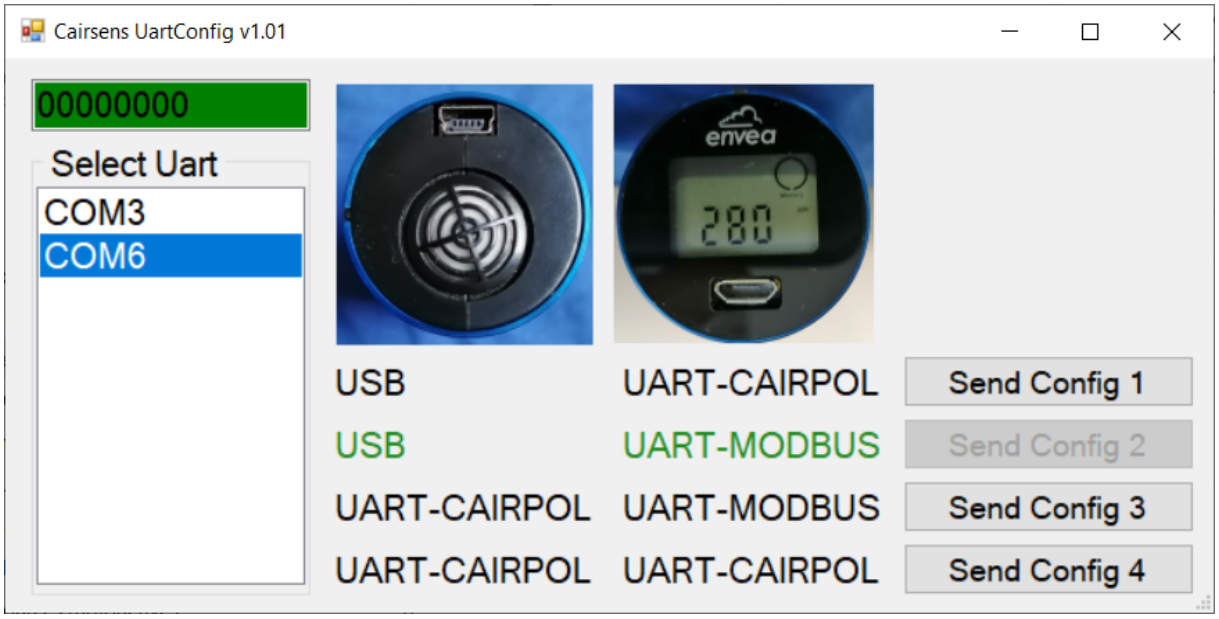
6. La configuration peut alors être modifiée. Pour cela, il faut choisir la configuration souhaitée et cliquer sur le bouton directement à sa droite. Par exemple, dans le cas considéré ici, le passage de la configuration UART-CAIRPOL vers la configuration UART-MODBUS se fait en cliquant soit sur « Send Config 2 », soit sur « Send Config 3 ».

Lorsque l'opération est terminée, la cellule en haut à gauche de la fenêtre affiche le message « No Answer » et devient rouge car la configuration de départ n'est plus active.



7. Pour vérifier que le changement de configuration s'est bien passé, il est possible de faire le test suivant :

- Fermer la fenêtre
- Exécuter à nouveau uartconfig.exe
- Dans la première fenêtre, choisir la configuration qui vient d'être chargée
- Sélectionner le port de COM sur lequel est branché le capteur
- Vérifier que le capteur est bien détecté et configuré comme souhaité. Dans le cas considéré ici, le changement de configuration de UART-CAIRPOL vers UART-MODBUS s'est bien passé, et la configuration UART-MODBUS apparait bien en vert dans la colonne de droite :



4 DEFAUTS DE FONCTIONNEMENTS

Les défauts détectables et affichés à l'écran sont :

Message affiché à l'écran	**Description du défaut**
NOTR 	**NOT READY** Le capteur est en préchauffage. Cet affichage apparait au démarrage.
LASE	**LASER ERROR** Contacter ENVEA par email à info@cairpol.com.
HEAT	**HEAT ERROR** L'humidité relative reste supérieure à 60% pendant plus de 10 minutes. Le capteur peut toujours envoyer des données mais elles risquent d'être moins précises.
TRH	**TRH ERROR** Les mesures de température et humidité du capteur sont hors spécifications. Le capteur peut toujours envoyer des données mais elles riquent d'être moins précises. Cette erreur peut apparaître pendant le préchauffage.
FAN	**FAN ERROR** La vitesse du ventilateur est hors tolérences mais le ventilateur fonctionne toujours. Le capteur peut toujours envoyer des données mais elles riquent d'être moins précises. Cette erreur peut apparaître pendant le préchauffage.
MEM	**MEMORY ERROR** Le capteur ne peut pas accéder à sa mémoire, certaines fonctions intelligentes internes ne seront pas disponibles. Le capteur peut toujours envoyer des données mais elles riquent d'être moins précises.
SLEE	**SLEEP STATE** Le laser, le ventilateur et le chauffage sont éteints.
CAL	**FIN DE VIE**

Message affiché à l'écran	**Description du défaut**
	Le capteur a atteint le terme de sa durée de vie d'1 an. Ce délai dépassé, les performances métrologiques du CAIRSENS PM ne sont plus garanties. Le laser risque de ne plus fonctionner. Il est conseillé de renouveler le CAIRSENS PM rapidement (contacter ENVEA par email à info@cairpol.com).
	**MEMOIRE PLEINE** Le capteur continue à fonctionner normalement, mais il enregistre les nouvelles mesures en écrasant les mesures les plus anciennes (sur l'écran montré ici, il mesure 0). L'opérateur doit extraire régulièrement les données avant d'atteindre la capacité maximale de stockage du capteur pour éviter de perdre des mesures.

5 MAINTENANCE DU CAIRSENS PM

5.1 CONSIGNES DE SECURITE

Les consignes de sécurité doivent être respectées à tout moment par l'utilisateur.

- Couper l'alimentation électrique lorsque vous effectuez la maintenance du capteur CAIRSENS PM.
- Le personnel doit être convenablement formé au bon fonctionnement du capteur avant de commencer à le faire fonctionner.
- Utiliser seulement les accessoires fournis.
- Ne pas obturer le l'entrée d'air, ni l'orifice d'extraction d'air.
- Ne pas tenir le CAIRSENS PM dans les mains pendant les mesures.
- Ne pas utiliser dans un milieu poussiéreux, corrosif, explosif, dans un environnement en présence d'autres gaz (gaz de combustion, solvant, chlore, vapeurs acides et basiques…).
- Respecter les conditions d'utilisation (cf. Caractéristiques techniques).

En ce qui concerne la sécurité, le fabricant ne peut être tenu responsable des conséquences résultant de:

- L'utilisation de l'appareil par du personnel non qualifié,
- L'utilisation de l'appareil dans des conditions autres que celles spécifiées dans le présent document,
- L'utilisation de pièces de rechange ou d'accessoires non fournis par ENVEA.
- L'utilisation de cet appareil d'une manière qui n'est pas approuvé par ENVEA est déconseillée et peut causer des dommages au personnel utilisateur et au matériel. La non-utilisation de pièces de rechange spécifiques peut réduire l'efficacité du dispositif de sécurité.
- La modification de l'appareil par l'utilisateur,
- Le non-entretien de l'appareil.

5.2 OPÉRATION DE MAINTENANCE

Le CAIRSENS PM est sans maintenance. Il doit être remplacé annuellement.

6 ANNEXES

6.1 PROTOCOLE MODBUS

6.2 PROTOCOLE CAIRSENS PM UART

6.3 PLAN

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;
```

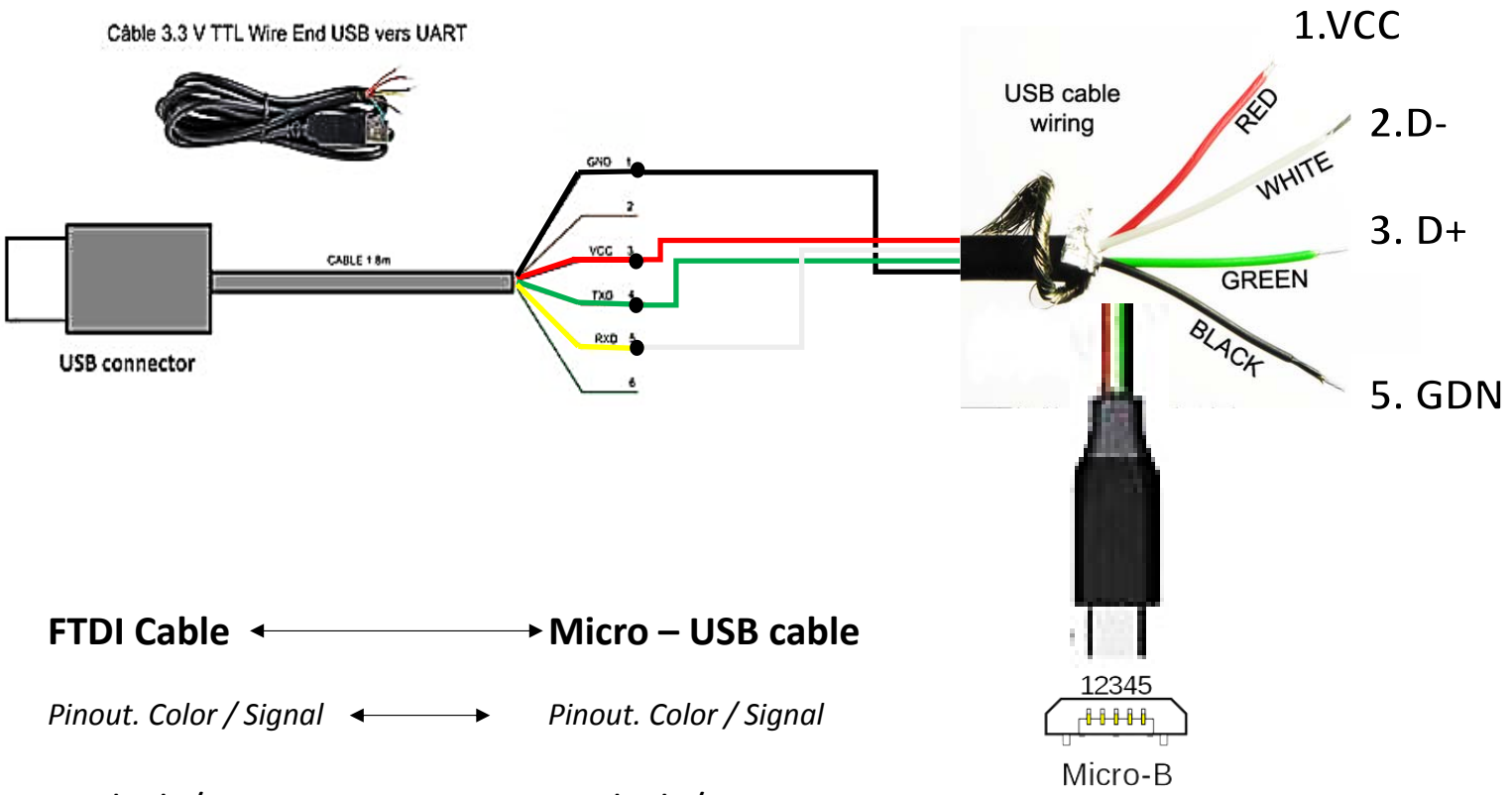
```
	do {

		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-