**Protocole lorawan projetcns**

**Protocole inspiré de cayenne**

# **Discussion générale**

Le format des données capteurs envoyés par un nœud lora. [Spécification]

Le format de données mis en place cherche à réduire la taille des données transmises, tout en étant générique pour supporter une grande variété de capteurs.

Les données sont codées en binaire pour plus de compacité, en orientation petit-boutiste (little-endian bit-poids-faible-en-premier)

Le format proposé n’est pas d’une très grande régularité ou simplicité, en particulier il n’est pas composé d’un motif fixe reproduit à l’identique pour toutes les données, comme c’est le cas par exemple pour le format Cayenne qui a servi de source d’inspiration : [Cayenne Docs (mydevices.com)](https://developers.mydevices.com/cayenne/docs/lora/)

([https://http://mydevices.com/cayenne/docs/lora/](https://http:/mydevices.com/cayenne/docs/lora/)).

Le but est de réduire la taille des données transmises.

Le nombre d’octets d’un champ peut varier et certaines valeurs peuvent « déborder » de leur cadre (ont une portée globale). [pour être plus clair sur ce dernier point j’évoquerais le cas de l’heure (timestamp)]

Toutes les mesures de capteurs sont horodatées.

Il se peut que plusieurs mesures aient une même date, parce qu’elles proviennent d’un même capteur ou encore parce que nous avons reçu simultanément des mesures de plusieurs capteurs par exemple.

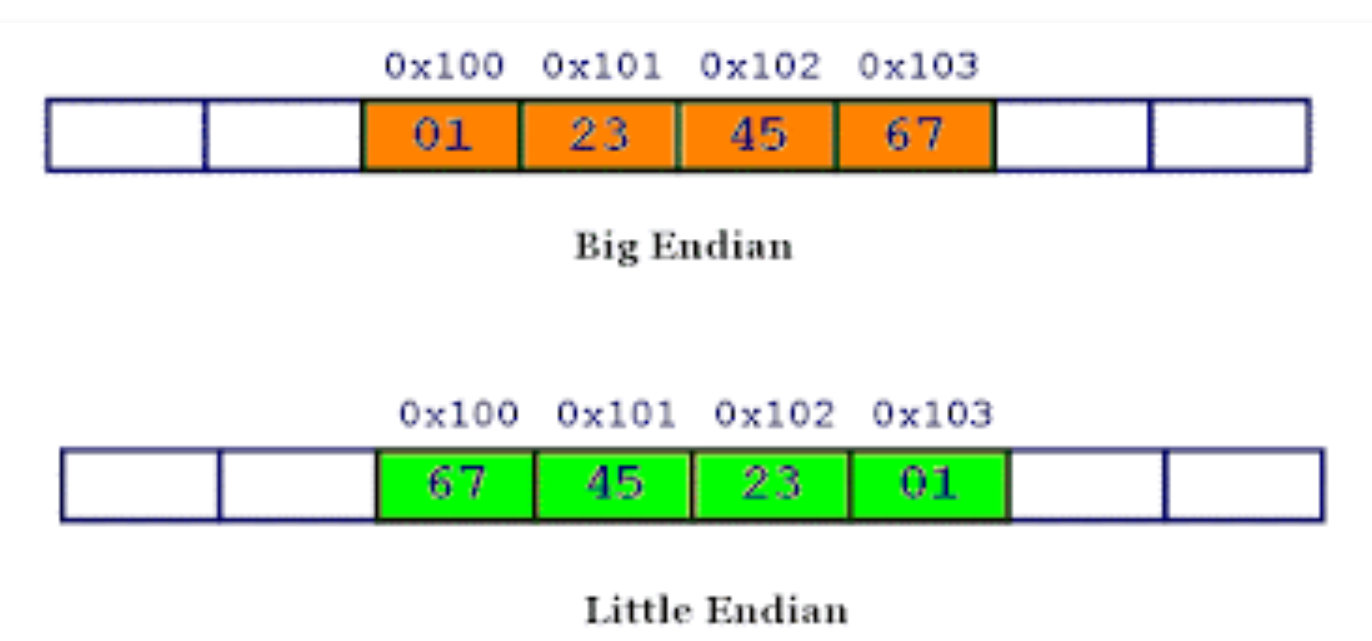
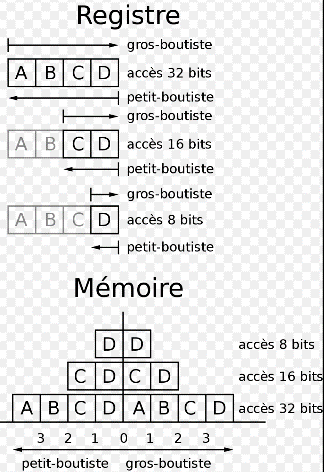
La répétition à l’identique de données dans une même trame représente une perte de bande passante.

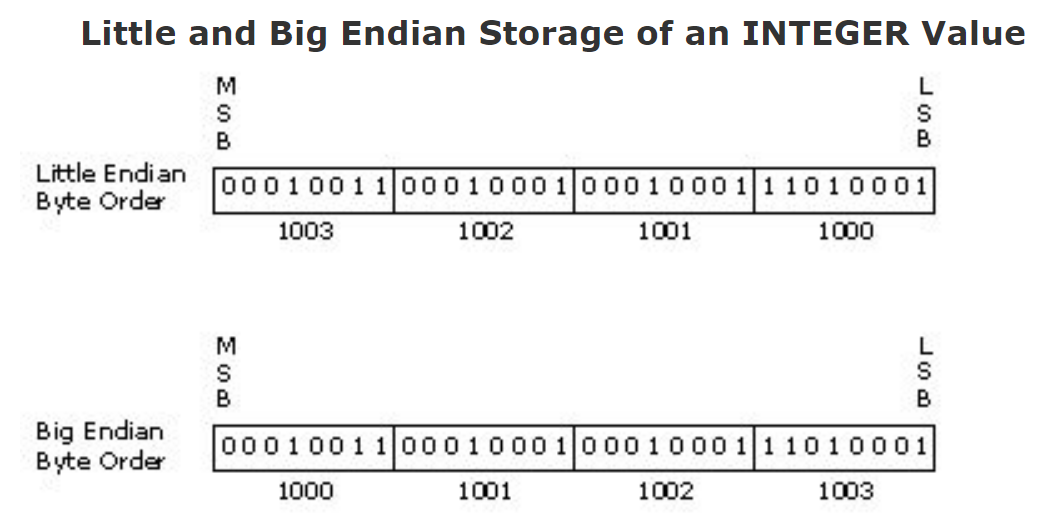
Aussi, dans le format proposé ici, les dates d’horodatage (timestamp) sont valides jusqu’à la fin de la trame de données envoyée, ou jusqu’à ce qu’un nouveau timestamp soit défini.

Toutes les valeurs de mesure transmises ont donc un timestamp qui est celui en cours de validité, qu’il ait été défini juste avant elles ou bien au début de la trame.

MSB big-endian gros-boutiste démarre avec les octets de poids fort (TCPIP,PCIexpr)

LSB little-endian petit-boutiste démarre inverse avec les octets de poids faible





# **Glossaire**

. **Nœud** : Organe de collecte et de transmission des données capteurs, un nœud peut être connecté à plusieurs capteurs.

. **Data Channel** : Et une source de données, Le plus logique est de faire coïncider une source de données avec une source physique, [ un capteur ou le nœud]. Mais ce n’est pas une obligation non plus, un unique capteur physique pourrait être la source de deux, ou plus, *Data Channel* si cette configuration à un sens pour votre application. Les *Data Channel* sont identifiés par un numéro positif et le numéro 0 est réservé pour le nœud.

. **Data Type** : Est un identifiant de type de données. La liste des identifiants est, mais elle peut être allongée pour y ajouter des types de données non encore identifiés. Permet de savoir si les données qui suivent représentent une température, une position géographique, une pression, … . Il permet également de connaître la taille des données qui le suivent.

# **Structure des trames montantes**

Il s’agit des trames de données envoyées par le nœud vers la passerelle ou vers le serveur.

La trame est composée d’un octet de format/version, suivi d’une répétition du motif suivant ( N est toujours ≥ 1) :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1 octet** | **1 octet** | **N octets** | **N octets** | **N octets** | **N octets** | **...** |
| Format/version | Data Channel n | Data Type X | données | Data Type Y | données |  |

Où le nombre de couples (*Data Type*, données) est au minimum de 1.

Exemple :

Supposons que nous souhaitions envoyer les données de deux capteurs, chacun étant identifié comme une source de données.

Le capteur 1 est le Data Channel 1. (Pression + Température)

Le capteur 2 est le Data Channel 2. (Humidité)

Chacune de ces valeurs est associée à un *Data Type* différent.

DataTypePress le *Data Type* qui permet d’identifier une mesure de pression, disons que la mesure est codée sur deux octets. DataTypeTemp pour la température codée sur un octet.

DataTypeHumi à l’humidité codée sur un octet.

Nous supposerons également que DataTypePress et DataTypeHumi sont codés sur un octet tandis que DataTypeTemp requière deux octets.

Nous obtenons alors la trame suivante :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1 octet** | **1 octet** | **1 octet** | **2 octets** | **2 octets** | **1 octet** | **1 octet** | **1 octet** | **1 octet** |
| 0xC1 | Data Channel 1 | DataTypePress | pression | DataTypeTemp | température | Data Channel 2 | DataTypeHumi | humidité |

Le nombre d’octets utilisés par les *Data Type* et par la valeur de mesure associée est défini par la liste des *Data Type* qui nous étudierons plus loin.

De même, le nombre de couples (*Data Type*, données) est déduit du *Data Channel* comme nous le verrons sous peu.

## **Le champ de format/version**

Il est codé sur un octet et sa valeur est différente de 0x00. Sa structure est la suivante :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Data Channel*** | | | | | | | |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Format | | | Version | | | | |

La valeur de Format est obligatoirement 0x6 (0b110). La Version qui correspond à ce document est la 0x01.

## **Le champ Data Channel**

Il permet d’identifier une source de données et de savoir combien de couples (*Data Type*, données) suivront.

Il est codé sur un octet dont le découpage est le suivant :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Data Channel*** | | | | | | | | |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| ClearGlobal | DataChannelId | | | | NbData | | | |

DataChannelId : est l’identifiant de la source de données à proprement parler sur quatre bits, où le bit 3 de l’octet est celui de poids le plus faible, ce qui autorise 16 identifiants. L’identifiant 0 est réservé pour le nœud, cela nous laisse donc la possibilité d’identifier 15 capteurs.

NbData : indique le nombre de couples (*Data Type*, données) à suivre. Il est codé sur 3 bits, où le bit 0 est celui de poids le plus faible, ce qui laisse donc la possibilité d’avoir jusqu’à 7 couples (la valeur 0 ne peut pas être utilisée et est réservée pour d’éventuels usages futurs).

ClearGlobal : indique si le contexte global doit être vidé ou non. Si ce bit est à 1 alors toutes les valeurs globales courantes n’ont plus cours, nous repartons d’un contexte global vide. Si ce bit est à 0 alors le contexte global courant est toujours d’actualité.

## **Le champ *Data Type***

Sa longueur minimale est d’un octet et il n’a pas de longueur maximale, même s’il y a peu de chances qu’il dépasse 2 octets un jour. Il est composé d’octets au format identique :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Data Type*, octet x** | | | | | | | |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| More | IdPart | | | | | | |

More : Ce bit (7) indique s’il faut un octet supplémentaire pour avoir l’identifiant *Data Type* complet. Ainsi, si l’identifiant est inférieur ou égal à 127, alors un octet, avec le bit More à 0 est suffisant 127 = 0111 1111.

Sinon, le bit More est positionné à 1, et un octet supplémentaire contenant les bits 7 à 13 du *Data Type,* et avec More à 0, est nécessaire et suit l’octet courant. Et ainsi de suite si deux octets ne sont pas suffisants. 0000 0000 1000 0000

IdPart : Partie de l’identifiant *Data Type*. Souvenez-vous que les données sont transmises en petit-boutisme (les octets de poids faible en premier).

# **Les différents Data Type et les données associées**

Certaines valeurs peuvent avoir une portée globale, c’est-à-dire que leur valeur est applicable à tous les capteurs et aux valeurs qu’ils produisent.

Ce cas a été illustré dans le premier chapitre avec le cas de l’horodatage des données des capteurs.

*A priori*, seul le *Data Channel* 0, celui du nœud, produit des données à valeur globale.

La possibilité pour un *Data Type* d’avoir une portée globale est indiquée dans les descriptions à suivre, ainsi que les conditions où cette portée globale est effective.

Voici une liste des identifiants *Data Type* et la taille et le codage des données associées. Ceci permettra de mettre en évidence l’écriture du code :

**SPECIFICATIONS DES TYPES DE DONNEES**

## TimestampUTC (0x00)

|  |  |
| --- | --- |
| **TimestampUTC** | |
| Est une valeur d’horodatage (timestamp), exprimée en heure universelle.  Est un entier non signé qui compte le nombre de secondes écoulées depuis le 1er janvier 2000 à minuit. | |
| DataTypeId : | 0x00 |
| Portée globale : | Oui, si présent dans le flux du *Data Channel* 0 |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 4 octets (uint32\_t LSB) |

## BattVoltageMV (0x01)

|  |  |
| --- | --- |
| **BattVoltageMV** | |
| Est la tension batterie, en millivolts. Codée avec un entier non signé de 16 bits.  Peut exprimer une tension batterie comprise entre 0 V et 65,536 V, au millivolt près. | |
| DataTypeId : | 0x01 |
| Portée globale : | Oui, si présent dans le flux du *Data Channel* 0 |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 2 octets (uint16\_t LSB) |

## TempDegC (0x02)

|  |  |
| --- | --- |
| **TempDegC** | |
| Une température, en degrés Celsius. Codée avec un entier non signé sur 16 bits et une résolution de 0,1 °C par bit sur la plage de -273,1 à 1 365,1 °C. La valeur -273,2 °C, indique une erreur.  Les données sont codées de la façon suivante :   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octet 0** | **Octet 1** | | | | | | | | | 7-0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | ValueLSB | AlarmH | AlarmL | ValueMSB | | | | | |   Où :   * La valeur est : Octet\_0 | (( Octet\_1 & 0x3F ) << 8) * AlarmL indique si l’alarme de niveau bas est active (1) ou non (0). * AlarmH indique si l’alarme de niveau haut est active (1) ou non (0).   La température vaut : valeur / 10 – 273,2. | |
| DataTypeId : | 0x02 |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 2 octets (uint16\_t LSB) |

## VoltageV (0x0C)

|  |  |
| --- | --- |
| **VoltageV** | |
| Une tension codée au moyen d’un entier signé sur 16 bits, où chaque bit représente 0,01 V. Peut donc représenter une tension comprise entre -327,68 et 327,67 V.  La tension : valeur x 0,01. | |
| DataTypeId : | 0x0C |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets (int16\_t LSB) |

## 4.5 Acceleration3DG (0x0D)

|  |  |
| --- | --- |
| **Acceleration3DG** | |
| Une accélération avec ses trois composantes (X ; Y ; Z). Chaque composante est codée au moyen d’un entier signé sur 16 bits où chaque bit correspond à 0,001 G.  Composante : valeur x 0,001. | |
| DataTypeId : | 0x0D |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets : X (int16\_t LSB), Y (int16\_t LSB), Z (int16\_t LSB) |

## 4.6 RainAmountMM (0x0E)

|  |  |
| --- | --- |
| **RainAmountMM** | |
| La quantité de pluie tombée dans un intervalle de temps.  L’intervalle de temps est le premier codé dans la trame, en secondes, au moyen d’un entier non signé de 24 bits.  La valeur, avec une précision de 0,1 millimètres, est représentée par un entier non signé codé sur deux octets de la façon suivante :   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octets de la valeur** | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | ValueLSB | | | | | | | | Res | Alrm | ValueMSB | | | | | |   Où :   * La valeur vaut : octet\_0 | ((octet\_1 & 0x3F) << 8) * Alrm, si positionné à 1 alors indique que la valeur est dans la zone d’alarme. * Res est un bit réservé pour de futurs usages.   La quantité de pluie : valeur x 0,1. | |
| DataTypeId : | 0x0E |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets : intervalleSec (uint24\_t LSB), quantitéPluie (uint16\_t LSB) |

## 4.7 BattVoltageFlagsMV (0x0F)

|  |  |
| --- | --- |
| **BattVoltageFlagsMV** | |
| Est la tension batterie, en multiples de 10 millivolts, avec des indications d’état.  Peut exprimer une tension batterie comprise entre 0 V et 40,96 V, avec un pas de 10 mV.  Codée avec un entier non signé de 16 bits de la façon suivante :   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octets** | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | ValueLSB | | | | | | | | Réservé | Réservé | Réservé | LowBatt | ValueMSB | | | |   Où :   * La valeur vaut : Octet\_0 | ((octet\_1 & 0x0F) << 8) * LowBatt est positionné à 1 si la tension batterie est tombée en dessous de son seuil de niveau bas. * Les bits 5 à 7 du second octet sont réservés pour de futurs usages.   La tension batterie vaut : valeur x 10 mV. | |
| DataTypeId : | 0x0F |
| Portée globale : | Oui, si présent dans le flux du *Data Channel* 0 |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets (uint16\_t LSB) |

## 4.8 SoilMoistureCb (0x10)

|  |  |
| --- | --- |
| **SoilMoistureCb** | |
| Est l’humidité du sol exprimée en centibars, avec une précision de 1 centibar sur la plage 0 à 254 centibars, et à une profondeur donnée, avec une précision de 1 centimètre sur la plage 1 à 511 centimètres.  Ce Data Type peut contenir plusieurs mesures si un capteur mesure l’humidité du sol à plusieurs profondeurs.  Les données sont composées d’une répétition, 1 fois ou plus, du même motif de format de données. Le motif de base (le format de base) se décline en deux versions : une courte et une longue.  La version courte est utilisée lorsque la profondeur de mesure est inférieure ou égale à 63 centimètres et qu’il n’y a pas d’alarme à signaler. Elle est structurée de la façon suivante :   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octet 0** | | | | | | | | **Octet 1** | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | More | 0 | Depth | | | | | | Value | | | | | | | |   Où :   * More indique s’il y a encore un mesure après celle-ci (1) ou bien si elle est la dernière (0). * Depth est la profondeur de la mesure en centimètres. * Value est l’humidité en centibars.   Le format long est utilisé dans les autres cas de figure. Il est structuré de la façon suivante :   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octet 0** | | | | | | | | **Octet 1** | | | | | | | | **Octet 2** | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7-0 | | More | 1 | DepthLSB | | | | | | AlarmH | AlarmL | Res | Res | Res | DepthMSB | | | Value |   Où :   * More indique s’il y a encore un mesure après celle-ci (1) ou bien si elle est la dernière (0). * La profondeur Depth est : (Octet\_0 & 0x3F) | ((Octet\_1 & 0x07) << 6). * AlarmH est positionné à 1 pour signaler que la valeur est au-dessus du seuil haut d’alarme. * AlarmL est positionné à 1 pour signaler que la valeur est en dessous du seuil bas d’alarme. * Res sont des bits réservés pour de futurs usages.   La profondeur, en centimètres, vaut : Depth. Une valeur de 0 indique une erreur.  L’humidité, en centibars, vaut : Value. Une valeur de 255 indique une erreur. | |
| DataTypeId : | 0x10 |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 1. ou 3 octets) x n, avec n ≥ 1 |

## 4.9 PressurePa (0x14)

|  |  |
| --- | --- |
| **PressurePa** | |
| Une pression, en Pascals. Codée avec un entier non signé de 32 bits où chaque bit correspond à 0,1 Pa. Il est possible de représenter une pression comprise entre – 26 843 545,5 et 26 843 545,5 Pascals.  Le format des octets est le suivant :   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Octet 0** | **Octet 1** | **Octet 2** | **Octet 3** | | | | | | | | | 7-0 | 7-0 | 7-0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | ValueLSB0 | ValueLSB1 | ValueMSB0 | AlarmH | AlarmL | Abs | Sign | ValueMSB1 | | | |   Où :   * La valeur est : Octet\_0 | (Octet\_1 << 8) | (Octet\_2 << 16) | ((Octet\_3 & 0x0F) << 24) . * Le bit AlarmH est positionné à 1 si l’alarme haute est activée, il est à 0 sinon. * Le bit AlarmL est positionné à 1 si l’alarme basse est activée, il est à 0 sinon. * Le bit Abs indique si la valeur est absolue (1) ou si elle est compensée de la pression atmosphérique (0). * Le bit Sign indique si la valeur est négative (1) ou positive (0).   La pression vaut donc : Signe x valeur x 0,1 Pa. | |
| DataTypeId : | 0x14 |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets (uint32\_t LSB) |

## 4.10 Config (0x15)

|  |  |
| --- | --- |
| **Config** | |
| Permet de communiquer un élément de configuration d’un Data Channel.  Le Data Type est composé d’un identifiant de paramètre de configuration, d’un octet indiquant longueur de la valeur et de la valeur comme une chaîne de caractères. La chaîne n’est pas terminée par un caractère nul. Le nombre d’octets occupés par la chaîne est donc variable, mais ne peut pas dépasser 31 octets. Si la valeur est vide alors Len est à 0 et il n’y a pas d’octets après le second octet.  Le format des octets est le suivant :   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Octet 0** | | **Octet 1** | | **Octets 2..n** | | 7-6 | 5-0 | 7-5 | 4-0 |  | | Res | Id | Res | Len | Str |   Où :   * Id est l’identifiant de paramètre. La liste des valeurs possible est donnée plus bas. * Len est la longueur de la valeur, comprise entre 0 et 31. Elle correspond au nombre d’octets à suivre occupés par la valeur Str. * Str est la valeur, codée dans une chaîne de caractères. Ne se termine pas par nul. * Res sont des bits réservés pour de futurs usages. | |
| DataTypeId : | 0x15 |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 2 octets minimum, 33 octets au maximum. |

Voici la liste des identifiants de paramètres :

* 0x00 : n’est pas une valeur valide.
* 0x01 : Le nom.
* 0x02 : L’identifiant unique.
* 0x03 : Le type.
* 0x04 : Le numéro de version du micrologiciel (firmware).
* 0x05 : Le nom de l’expérience pour laquelle les données sont produites.
* 0x06 : La période de lecture d’un capteur, en secondes. La chaîne peut être convertie en un entier non signé de 32 bits. Elle peut être 0 si le capteur n’est lu que sur interruption.
* 0x07 : La période d’envoi de la configuration, en secondes. La chaîne peut être convertie en un entier non signé de 32 bits. Une valeur de 0 indique que la configuration n’est pas envoyée périodiquement, mais uniquement si un changement de configuration est détecté.
* 0x08 : La période de lecture du GPS, en secondes. La chaîne peut être convertie en un entier non signé de 32 bits. Une valeur de 0 indique que le GPS n’est pas utilisé.

## 4.11 RadioactivityBqM3 (0x1A)

|  |  |
| --- | --- |
| **RadioactivityBqM3** | |
| La mesure de la radioactivité, en Becquerels par mètres cube, avec une précision de 1 Bq/m3.  La longueur de ce Data Type est variable en fonction de la taille des données.  La valeur est composée d’un ou plusieurs octets, avec l’octet de poids faible en premier, tous formatés de la façon suivante :   |  |  | | --- | --- | | **Octet 0** | | | 7 | 6-0 | | More | ValuePart |   Où :   * ValuePart est une partie de la valeur. * More indique si encore au moins un octet supplémentaire est nécessaire pour encoder la valeur.   Ainsi, si le niveau de radiation est compris entre 0 et 127 Bq/m3, alors un seul octet est nécessaire. Pour un niveau entre 128 et 16 383 Bq/m3, deux octets sont nécessaires. Et ainsi de suite.  Le niveau de radiation (Bq/m3) est de : Octet\_0 & 0x7F | (Octet\_1 & 0x7F) << 7 | (Octet\_2 & 0x7F) << 14 | … | |
| DataTypeId : | 0x1A |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | Au moins un octet, cinq au maximum (LSB) |

## 4.12 HeaterStatus (0x1B)

|  |  |
| --- | --- |
| **HeaterStatus** | |
| Indique l’état d’un élément chauffant.  Cet état est codé sur un octet selon le format suivant :   |  |  | | --- | --- | | **Octet 0** | | | 7-1 | 0 | | Res | IsON |   Où :   * IsOn indique si l’élément chauffant est allumé (1) ou éteint (0). * Res sont des bits réservés pour des usages futurs. | |
| DataTypeId : | 0x1B |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octet |

## 4.13 SensorTypeMM3Hash32 (0x23)

|  |  |
| --- | --- |
| **SensorTypeMM3Hash32** | |
| La valeur hachée du nom du type de capteur au moyen de l’algorithme Murmur3, codée sur 32 bits. La graine utilisée est 0x434E5353 en petit boutisme. | |
| DataTypeId : | 0x23 |
| Portée globale : | Oui. Portée globale au DataChannel. |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets (uint32\_t LSB) |

## 4.14 ConfigMM3Hash32 (0x24)

|  |  |
| --- | --- |
| **ConfigMM3Hash32** | |
| La valeur hachée du fichier de configuration au moyen de l’algorithme Murmur3, codée sur 32 bits. La graine utilisée est 0x434E5353 en petit boutisme. | |
| DataTypeId : | 0x24 |
| Portée globale : | Oui. |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets (uint32\_t LSB) |

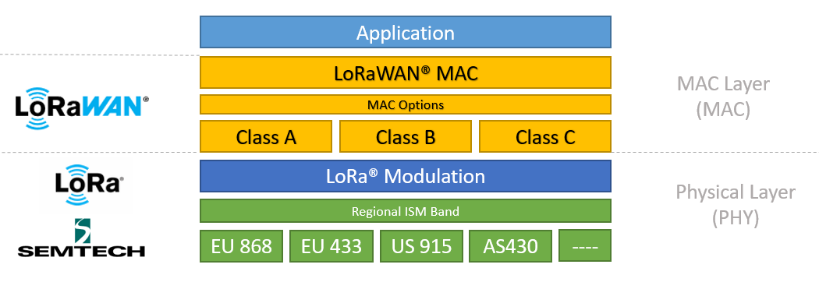
## 4.15 AppSoftwareVersionMMPHash32 (0x2B)

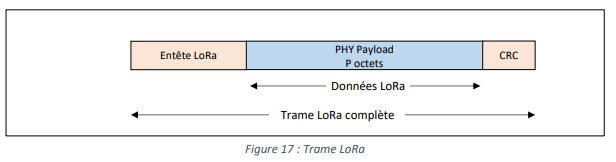
|  |  |
| --- | --- |
| **AppSoftwareVersionMMPHash32** | |
| Indique la version du logiciel applicatif. La version est composée d’une paire : (majeur.mineur.patch, hash 32 bits).  Le numéro de version majeur est ≤ 31, le mineur ≤ 63 et de patch ≤ 15.  Les données sont encodées de la façon suivante :   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **Octet 0** | | | | **Octet 1** | | | | | 7-4 | 3-0 | | | 7 | 6-2 | | 1-0 | | MinorLSB | Patch | | | Dev | Major | | MinorMSB | | **Octet 2** | | **Octet 3** | **Octet 4** | | | **Octet 5** | | | 7-0 | | 7-0 | 7-0 | | | 7-0 | | | hashB0 | | HashB1 | HashB2 | | | hashB4 | |   Le numéro de majeur est : (Octet\_1 & 0x7C) >> 2.  Le numéro de mineur est : ((Octet\_0 & 0xF0) >> 4) | ((Octet\_1 & 0x03) << 4).  Le numéro de patch est : Octet\_0 & 0x0F.  Si c’est une version de développement alors Dev (Octet\_1 & 0x80) est à 1, sinon est à zéro (release).  Le hash : Octet\_2 | (Octet\_3 << 8) | | (Octet\_4 << 16) | | (Octet\_5 << 24). | |
| DataTypeId : | 0x2B |
| Portée globale : | Oui, si présent dans le flux du *Data Channel* 0 |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets |

## 4.16 ResetSource (0x2C)

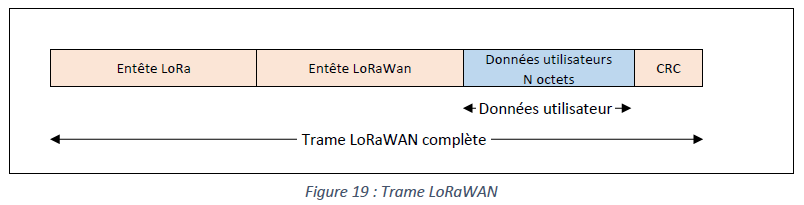
|  |  |
| --- | --- |
| **ResetSource** | |
| Indique la source, la cause, du dernier redémarrage du nœud.  La source est encodée au moyen d’un octet :   |  |  | | --- | --- | | **Octet 0** | | | 7-4 | 3-0 | | Res | Id |   Où :   * Res sont des bits réservés pour de futurs usages. * Id est l’identifiant de la source du redémarrage. La liste des identifiants possibles est la suivante :  |  |  |  | | --- | --- | --- | | **Id** | **Nom** | **Description** | | 0x0 |  | Réservé. | | 0x1 | BOR | Le nœud a été mis sous tension ou son alimentation est sortie des conditions de fonctionnement normales. | | 0x2 | resetPin | Le bouton « reset » du nœud a été activé. | | 0x3 | software | Le redémarrage fait suite à une demande du logiciel. | | 0x4 | softwareError | Le redémarrage fait suite à une demande du logiciel à cause de la détection d’une erreur. | | 0x5 | watchdog | Le chien de garde est en cause. | | 0x6 | independentWatchdog | Le chien de garde indépendant est en cause. | | 0x7 | lowPowerError | La mauvaise utilisation d’un mode basse consommation est en cause. | | 0x8 | optionBytesLoading | A redémarré pour charger les « option bytes » du microcontrôleur. | | 0x9 | firewall | La pare-feu a causé le redémarrage. | | 0xA-0xF |  | Réservés. | | |
| DataTypeId : | 0x2C |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | 1 octet |

# **Mise en œuvre**



Différence entre les trames

Lora c’est le type de modulation permettant d’envoyer des données entre un émetteur et un récepteur

Lorawan (avec un service supplémentaire) Architecture du réseau et format de trame. On parle de l’ensemble de la chaine de communication alors nous parlons de protocole LoRaWAN

*Le duty-cycle impose qu’un device Lora ne transmette pas plus de 1% du temps. 1% donc pour 1 temps je ne dois plus émettre pendant 99%. Cas Time on aire de 46.3ms le device doit émettre pendant 99\*46.3=4.58 secondes.*

# **6 Conception de la Trame**

Exemple avec le DataType température TempDegC DataTypeId : (0x02)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Octet 0** | **Octet 1** | | | | | | | |
| 7-0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ValueLSB | AlarmH | AlarmL | ValueMSB | | | | | |

Une température, en degrés Celsius. Codée avec un entier non signé sur 16 bits et une résolution de 0,1 °C par bit sur la plage de -273,1 à 1 365,1 °C. La valeur -273,2 °C, indique une erreur

Où : La valeur est : Octet\_0 | (( Octet\_1 & 0x3F ) << 8), AlarmL indique si l’alarme de niveau bas est active (1) ou non (0)0

AlarmH indique si l’alarme de niveau haut est active (1) ou non (0).

La température vaut : valeur / 10 – 273,2.

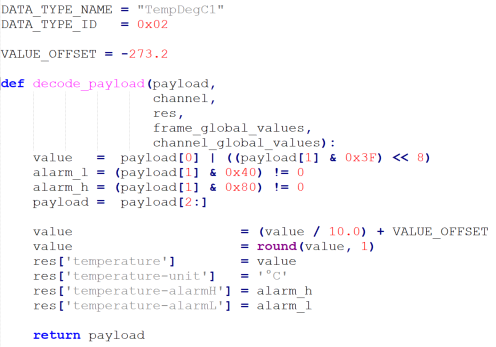
Portée globale : Non

Taille DataType : 1 octet

Taille des données : 2 octets (uint\_t LSB)

Device\_IOT\_stm32\_encode\_trame C++ Serveur\_ChirpStack\_decodage\_trame\_Python

drivers/Sensors/External

NameSensor{hpp ; cpp}

[cpp]

::writeDataToCNSSRFDataFrameSpecific(CNSSRFDataFrame \*pvFrame)

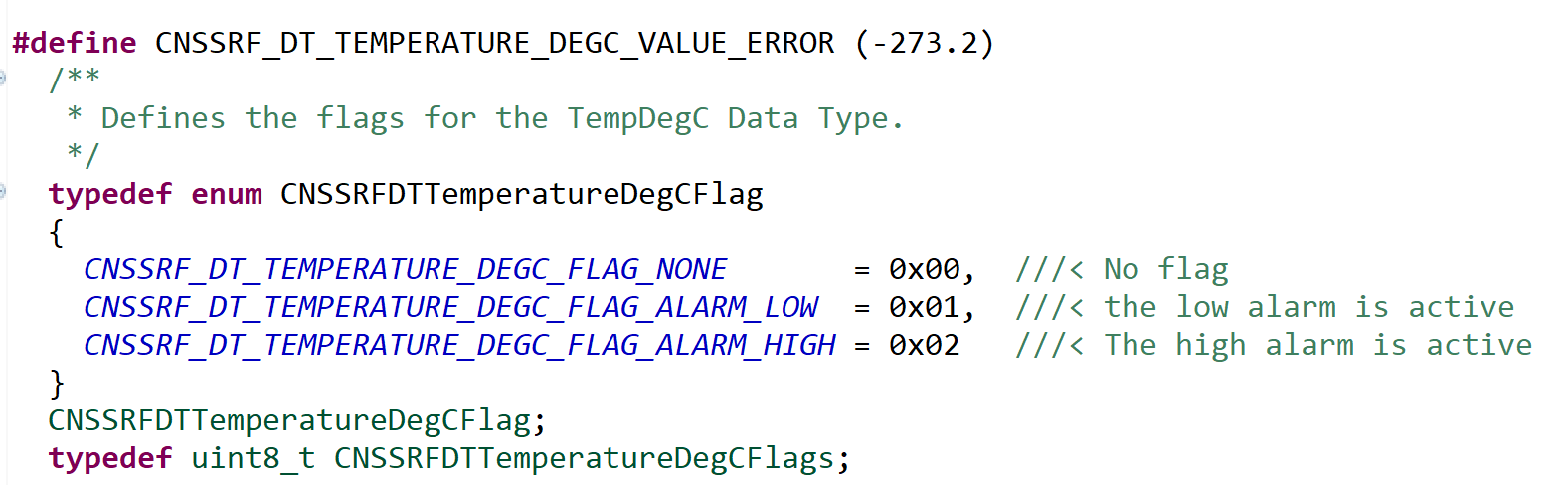
res &=

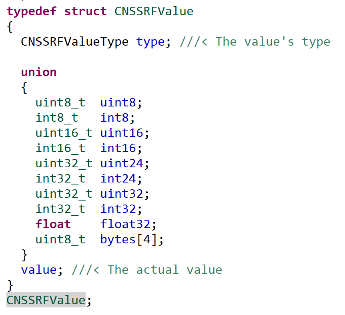
cnssrf\_dt\_temperature\_write\_degc\_to\_frame( pvFrame,

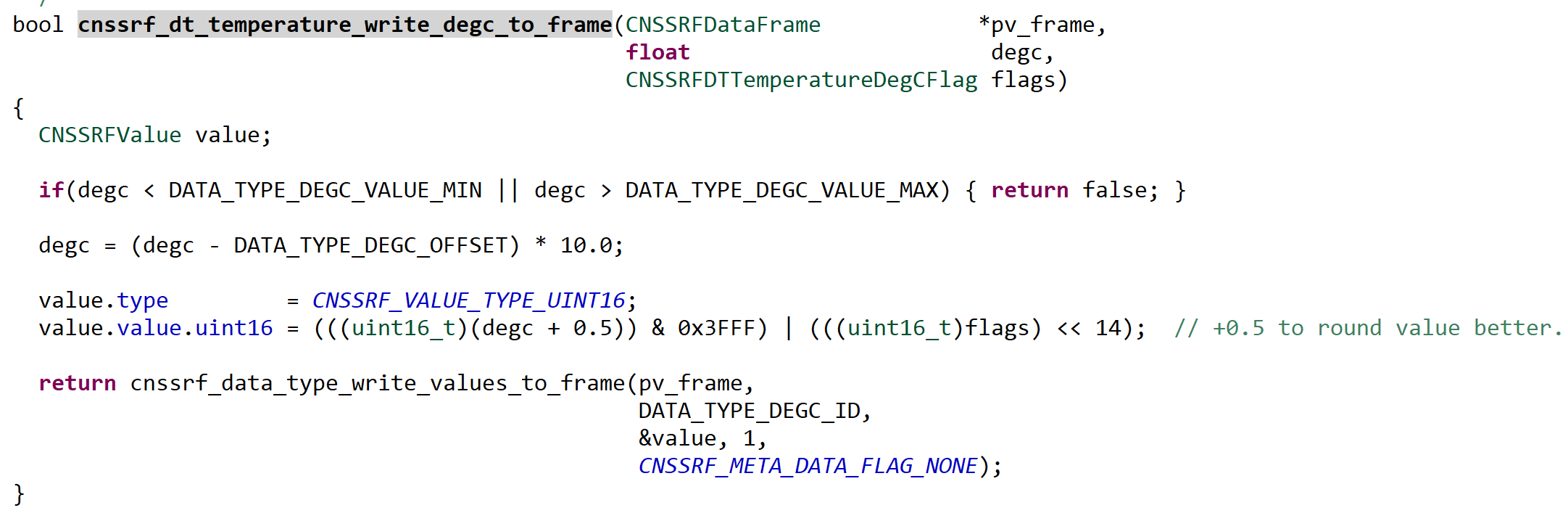
**this**->\_airTemperatureDegC, CNSSRF\_DT\_TEMPERATURE\_DEGC\_FLAG\_NONE**)**;

Codecs/connecsens-rf/datatypes/cnssrf-dt-temperature{h ;c}

[h]



[c]



----Les données sont lues depuis l’uart ensuite elles sont mises dans un buffer ensuite on extrait chaque donnée et la température est dans \_airTemperatureDegC.

Chaque données est ajoutés à la Trame avec cnssrf\_data\_type\_write\_values\_to\_frame (frame :cadre) mais juste avant on test la valeur et donne la mise en forme comme : par exemple 20.54 C°

Si 20.54 < (-273.2) ou 20.24 > 1365.1 alors retourne false

2937.4= (20.54 - (-273.2)) \* 10.0 ; //arrondie à 0.1 près pour 0.01 \*100

La valeurs est codé en 16 bits non signé : uint16 CNSSRFValueType value.Type et value.value.uint16 qu’on donne la valeur :

Cnssrf\_data\_type\_write\_values\_to\_frame( buffer (nombre de donnée, data\_channel courant, dataType…) 65535

identifiant du dataType 0x02

2937.4+0.5 = 2937.9 passage par référence des données capteur,

1011 0111 1001 = 2937 nombre de valeur qu’on veut envoyer

flags ici pas d’alarme = 0 = 0x00) ;

0.9 c’est 2\*0.9=1.8 2\*0.8=1.6 2\*0.6=1.2 2\*0.2=0.4 2\*0.4=0.8 2\*0.8=1.6 …. : 111000 11000 …

**1011 0111 1001 , 1110**00 11000 … Mais non on prend bien la partie entière

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Octet 0 LSB** | **Octet 1 MSB** | | | | | | | |
| 7-0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ValueLSB | AlarmH | AlarmL | ValueMSB | | | | | |

3 F F F

0011 1111 1111 1111

& 0000 **1011 0111 1001.**

0000 1011 0111 1001

| 0000 0000 0000 0000 <<14 lecture des alarme AlarmH et AlarmL décalage gauche de 14 pour lire le reste

0000 1011 0111 1001

Binaire nombre négatif

Un nombre peut s’exprimer en octet (le bit le plus fort fait le signe [0+, 1-])

Une variable sur 1 octet peut contenir -128 à +127 s’il autorise les négatifs pour le non signé de 0 à 255

Le nombre négatif d’un binaire (inverse les bits de ce nombre, on ajoute 1 car il n’y a qu’un seul zéro cela évite d’avoir un 0 positif et un zéro négatif pour le cas +0 différent de -0) c’est pour ça qu’on peut aller de -128 et non -127

116 = b0111 0100 et son négatif est (inverser non négatif + 1) = 1000 1011 +1= 1000 1100 (**complément à deux**)

Binaire nombre à virgule

La première méthode est la virgule fixe pour les microcontrôleurs comme 0110,1001 mais sur 0110,1000 on perd 3 bits inutilement.

La deuxième méthode virgule flottante signe \* mantisse \* base (exposant). La position de la virgule est fixée dans la mantisse et on la fait flotter en faisant varier l’exposant. IEEE 754 fixe le format

9,750 = b100111 la notion de virgule 1001,11 peut s’écrire 1001,11\*2exp0 ou 0,100111\*2exp(4)

Terme 100111 Mantisse et 100 Exposant donc 100111100 correspond à 1001,11 (b100=4) N=100111\*2exp4=1001,11

----------------------Conversion------------------------

B2->B10 : 110,1011 qui est 110= 2.2 2.1 2.0 , 1011= 2.-1 2.-2 2.-3 2.-4 = 5,

b110=6 et b1011= 2.-1+2.-3+2.-4=0.5+0.125+0.0625=0,6875 = 6,6875

B10->B2 : entière ok décimale : 0.6875 en binaire (multiplié par 2 et noté la partie entière successivement)

2\*0,6875 = 1,375

2\*0,375=0,75

2\*0.75 = 1,5

0,5\*2=1 =>partie entière : 1011 et donc 0,6875 = b0,1011

Virgule fixe : nombre de chiffre après la virgule est fixe

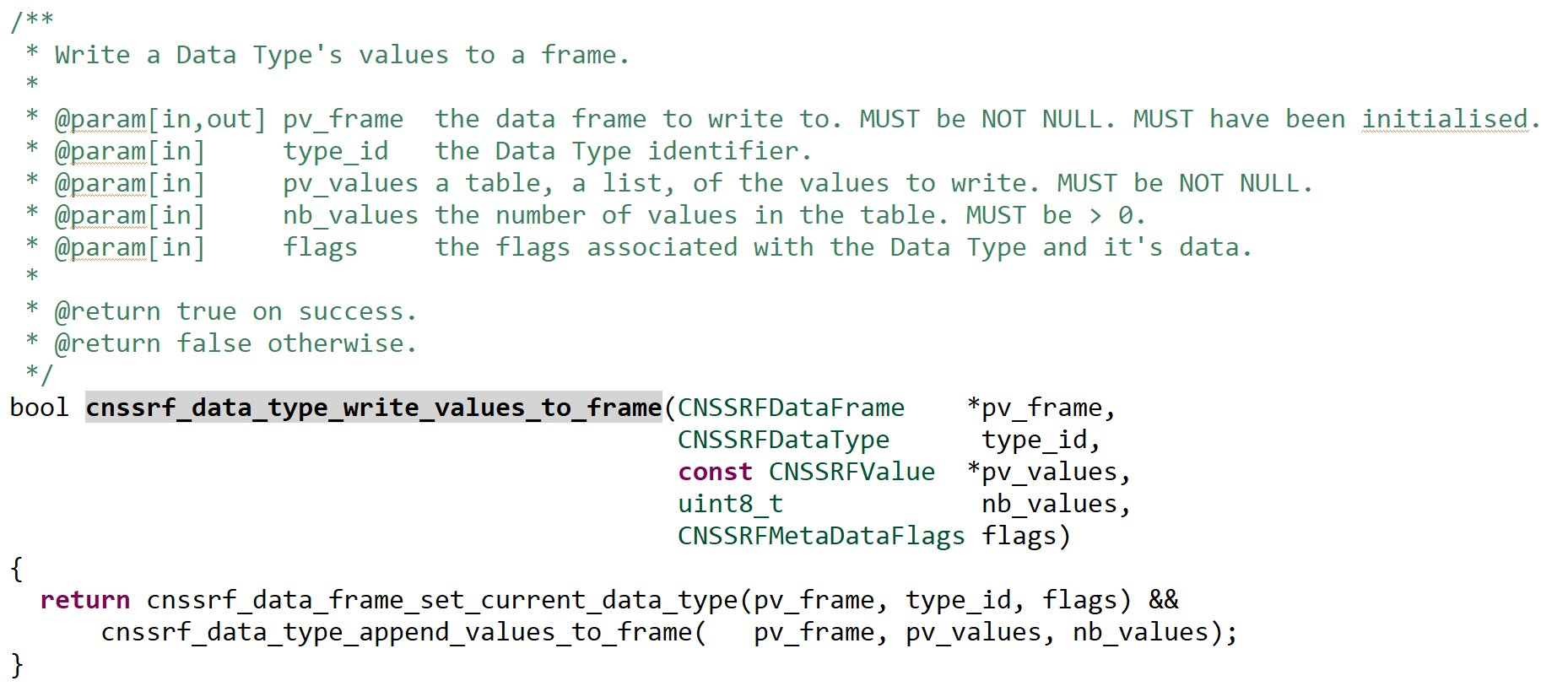
Virgule flottante : nombre de chiffre significatifs est fixe

2.-1=0.5=1/2 2.-2=0.25 2.-3=0.125 2.-4=0.0625

0.5+0.125=0.625

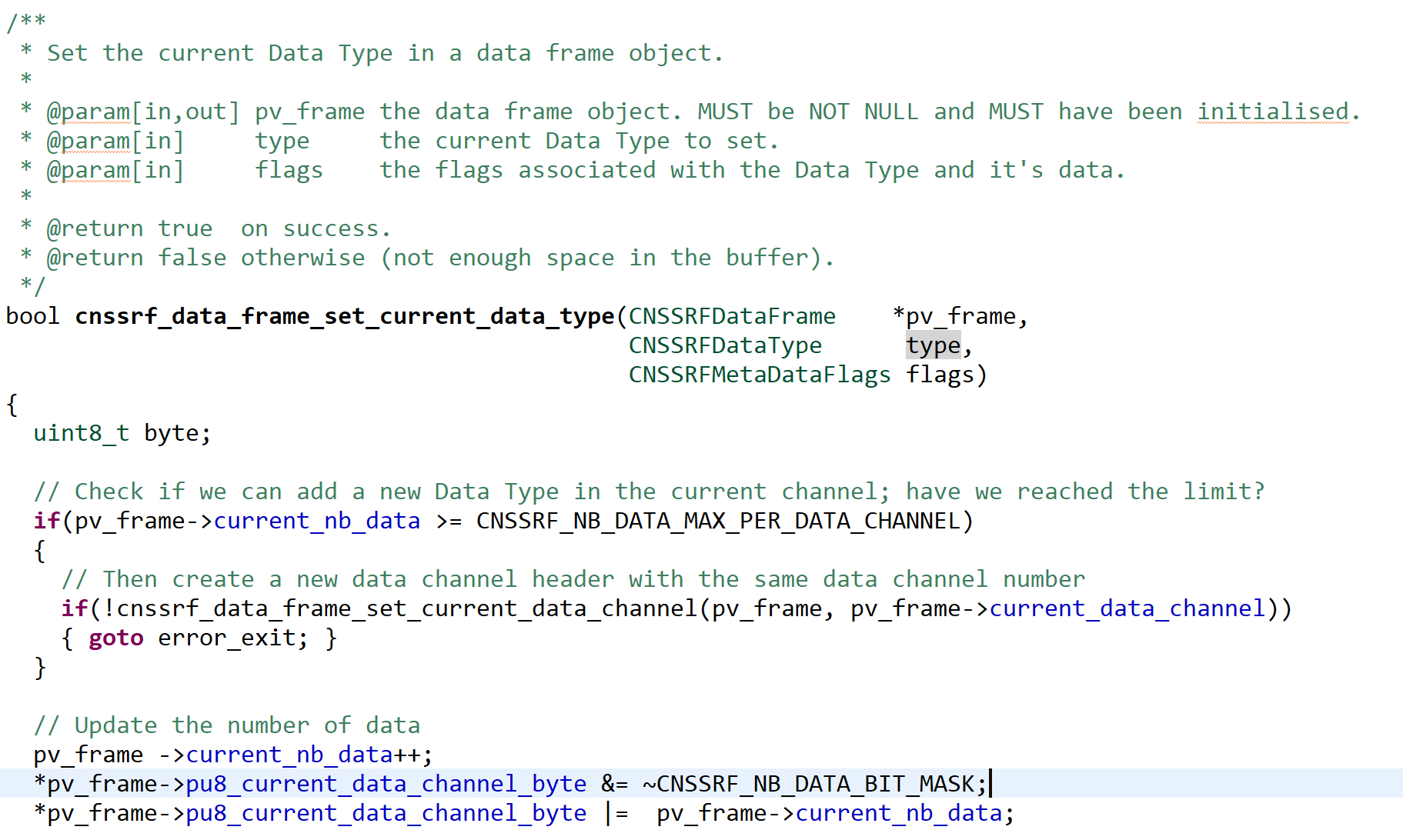
6.625=b110,101

La données part en écriture (datatypes.c)

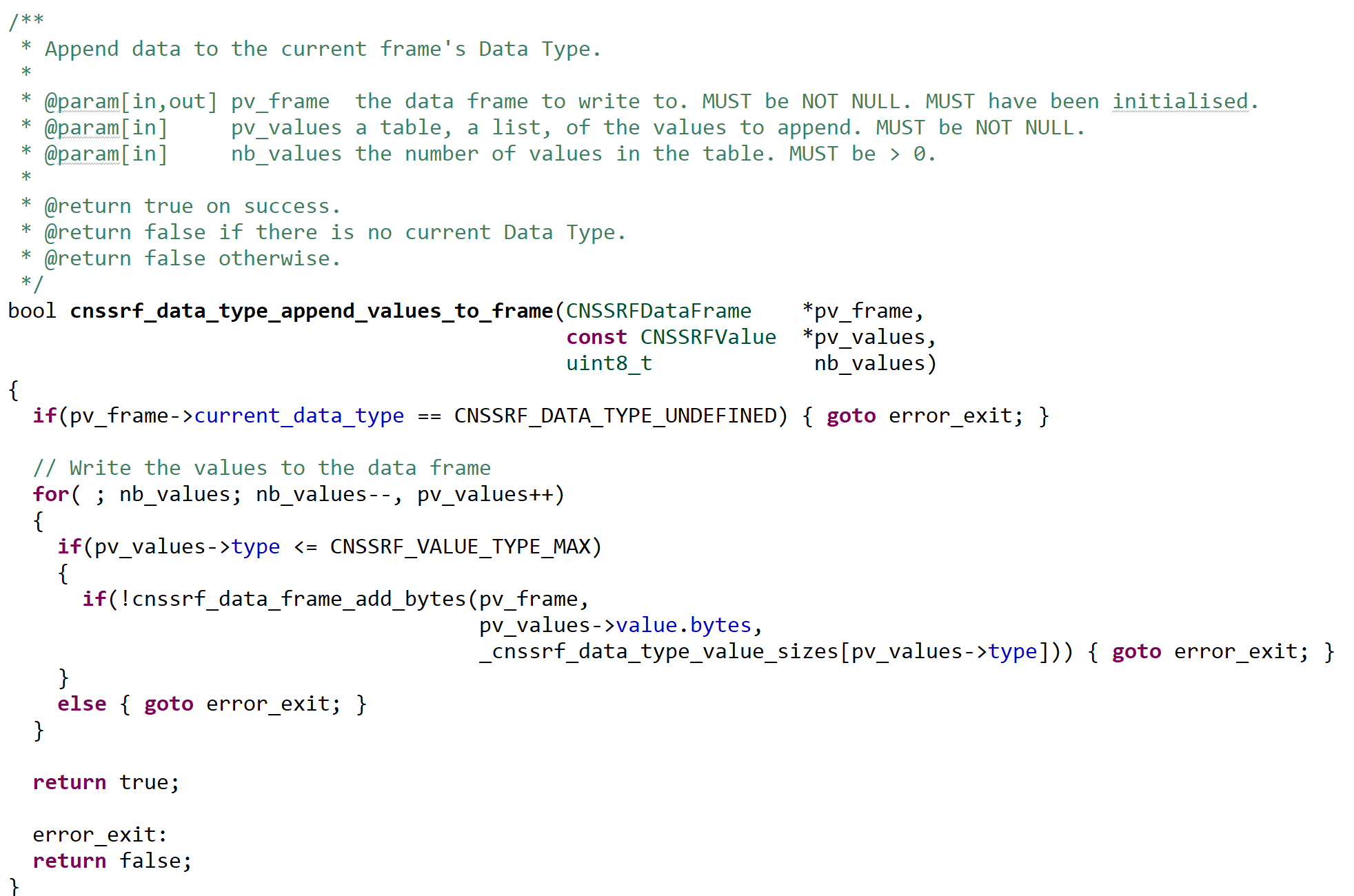


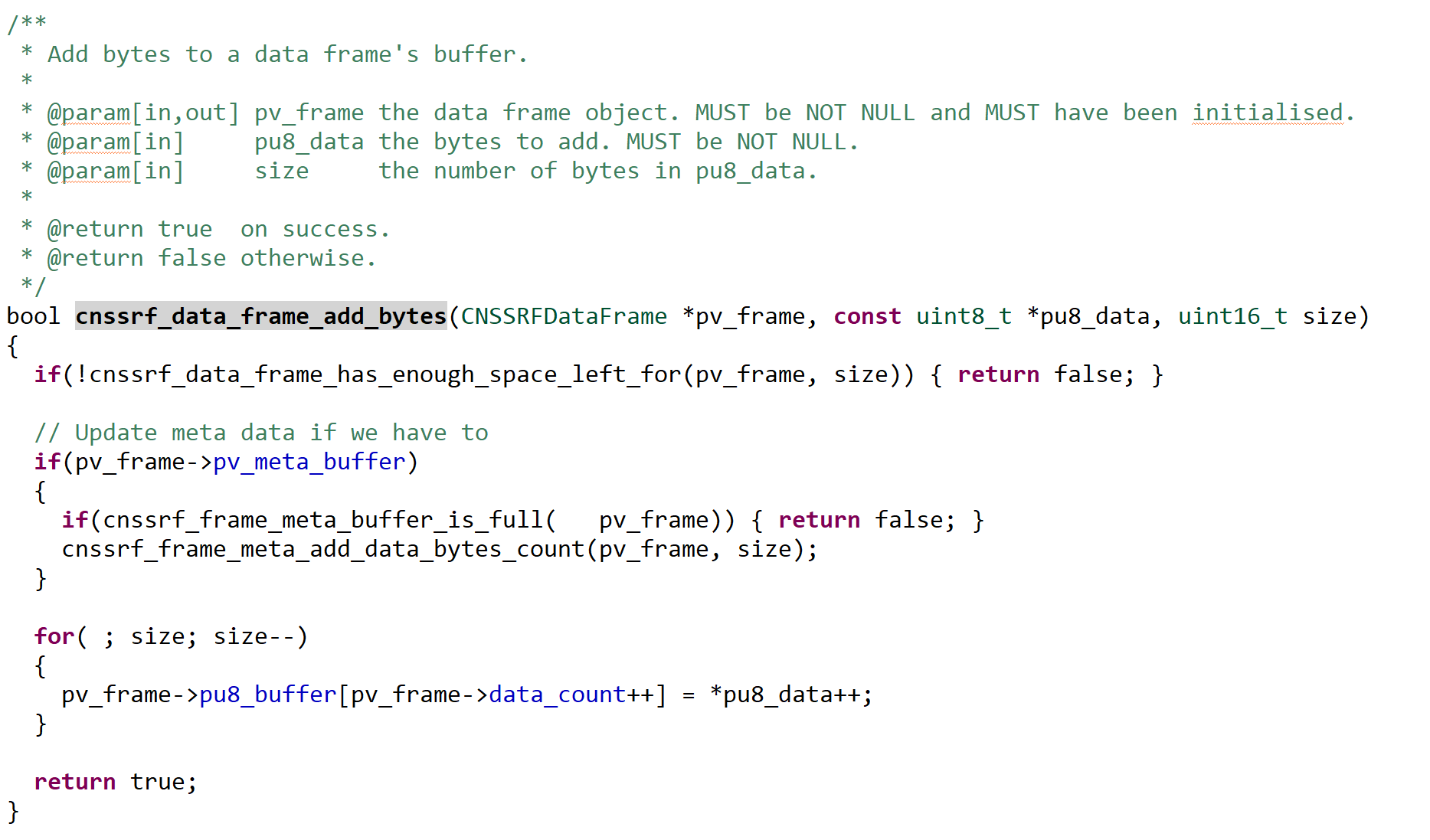
Ce qui nous amène à regarder les 2 autres fonctions

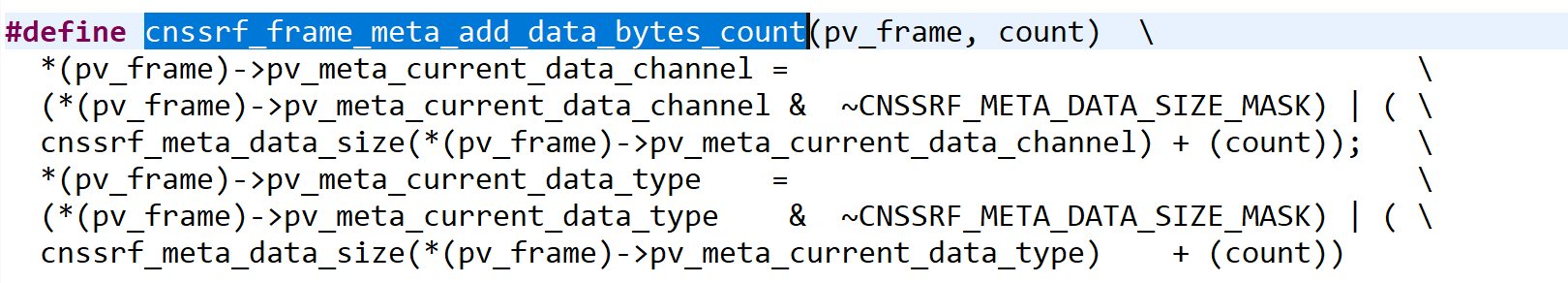
Création du datatype (dataframe.c)



….







Pour donner la trame

1 octet format (3bits)/version(5bits)

1 octet Data-Channel identifie une source de données et de savoir combien de couples (*Data Type*, données) suivront.

NbData le nombre de couple de (DataType/Données) qui suit (3bits)

DataChannelId (4bits) identifier au moins 15 capteurs source de données

ClearGlobal (1bits) contexte global vidé ou non à 0 alors le contexte global courant est toujours d’actualité.

Champ DataType au minimum (1 octet et pas de maximale même si 2 octets serait énorme)

IdPart identifiant le DataType (little-endian) et More (le bit7) s’il faut 1 octet supplémentaire pour avoir l’identifiant et on ajoute des octets si ce n’est pas suffisant.

Et nous avons les données du datatype

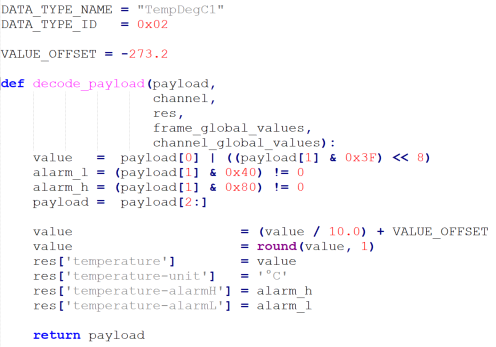
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1 octet** | **1 octet** | **1 octet** | **2 octets** | **2 octets** | **1 octet** | **1 octet** | **1 octet** | **1 octet** |
| 0xC1 | Data Channel 1 | DataTypeTemp  idPart 0x02 | Température  0000 1011 0111 1001 |  |  |  |  |  |

En définitif la construction de la trame Lora se fait dans le cnssrf-dataframe.c

Pour le décodage au niveau du serveur

Les données arrivent sur le serveur

{"applicationID": "8", "applicationName": "Test", "deviceName": "1263", "devEUI": "434e535306e31263", "rxInfo-gatewayID": "00800000a0001ad3", "rxInfo-uplinkID": "cd7ed5fd-ea02-46b8-851a-012c69a506d2", "rxInfo-name": "Multitech-Mesocentre", "rxInfo-rssi": -113, "rxInfo-loRaSNR": 4.8, "rxInfo-location-latitude": 0, "rxInfo-location-longitude": 0, "rxInfo-location-altitude": 0, "txInfo-frequency": 867900000, "txInfo-dr": 5, "adr": true, "fCnt": 11, "fPort": 2, "servertimestampUTC": "2022-03-07T14:49:35.387556", "data-DataChannel": 1, "data-node-timestampUTC": "2022-03-07T14:47:30", "data-node-batteryVoltage": 4.04, "data-node-batteryVoltage-unit": "V", "data-node-batteryVoltage-alarmIsLow": false, "data-CNSSRFConfigMM3Hash32": "E88A1445", "data-CNSSRFSensorTypeMM3Hash32": "D69A22D7", "data-temperature": 11.5, "data-temperature-unit": "°C", "data-temperature-alarmH": false, "data-temperature-alarmL": false, "data-CNSSRFDataTypeId": 2, "data-CNSSRFDataTypeName": "TempDegC"}





20.54 - (-273.2) \* 10 = 2937.4 (si \*100 = 29374+0.5=29374.5) -> 0111 0010 1011 1110

2937.4+0.5 = 2937.9 -> b 1011 0111 1001 , 1110 //arrondi à 0.5

3 F F F

0011 1111 1111 1111 0011 1111 1111 1111 (16263 maxi on déborde)

& 0000 **1011 0111 1001. & 0111 0010 1011 1110**

0000 1011 0111 1001 0011 0010 1011 1110

| 0000 0000 0000 0000 <<14 lecture des alarme AlarmH et AlarmL décalage gauche de 14 pour lire le reste

0000 1011 0111 1001 0011 0010 1011 1110

Octet1MSB Octet 0 LSB

Payload[1] 0000 1011 octet1 Payload[1] 0011 0010 octet1

Payload[0] 0111 1001 octet0 Payload[0] 1011 1110 octet0

Sur le serveur décodage :

Payload[0] 0111 1001 | ((Payload[1] 0000 1011 &0x3F )<<8) // <<8 lecture debut octet1 avec un & 0011 11111

0011 1111 0x3F 0011 1111 0x3F

&0011 0111 &0011 0010

======🡺 0000 1011 ======🡺 0011 0010

| 0111 1001 1011 1110 =>

======🡺 0000 **1011 0111 1001** = (2937/10) + (-273.2)= 20.5

**Les nombres négatifs fonctionne** nous pouvons avoir -20.82 ce qui peut donner

Complément à 2 de -4 c’est 00000100 => inverse 11111011 +1= 11111100 -4 - --4 - (-273.2) \* 10 = 2692

2772+0.5 = 2692.5 -> b 1010 1000 0100 , 1000 //arrondi à 0.5

3 F F F

0011 1111 1111 1111

& 0000 **1010 1000 0100.**

0000 1010 1000 0100

| 0000 0000 0000 0000 <<14 lecture des alarme AlarmH et AlarmL décalage gauche de 14 pour lire le reste

0000 1010 1000 0100

Octet1MSB Octet 0 LSB

Payload[1] 0000 1010 octet1

Payload[0] 1000 0100 octet0

Sur le serveur décodage :

Payload[0] 1000 0100| ((Payload[1] 0000 1010 &0x3F )<<8) // <<8 lecture debut octet1 avec un & 0011 11111

0011 1111 0x3F

&0000 1010

======🡺 0000 1010

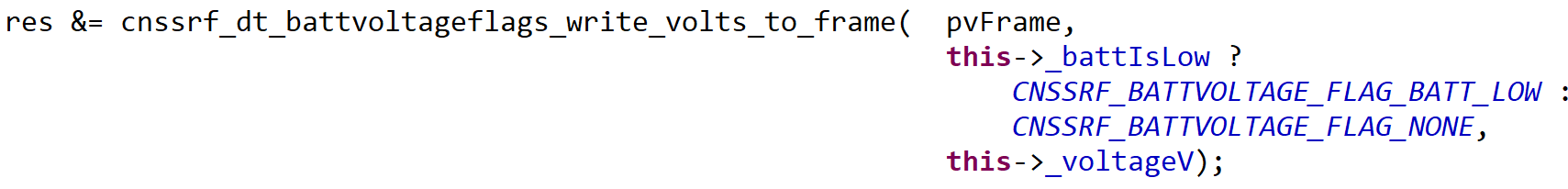
| 1000 0100

======🡺 0000 1010 1000 0100= (2692/10) + (-273.2)= -4

Pour vous aider de python lors de calcul comme ça.

Exemple avec le VoltageV ou BattVoltage DataTypeId : (0x02)

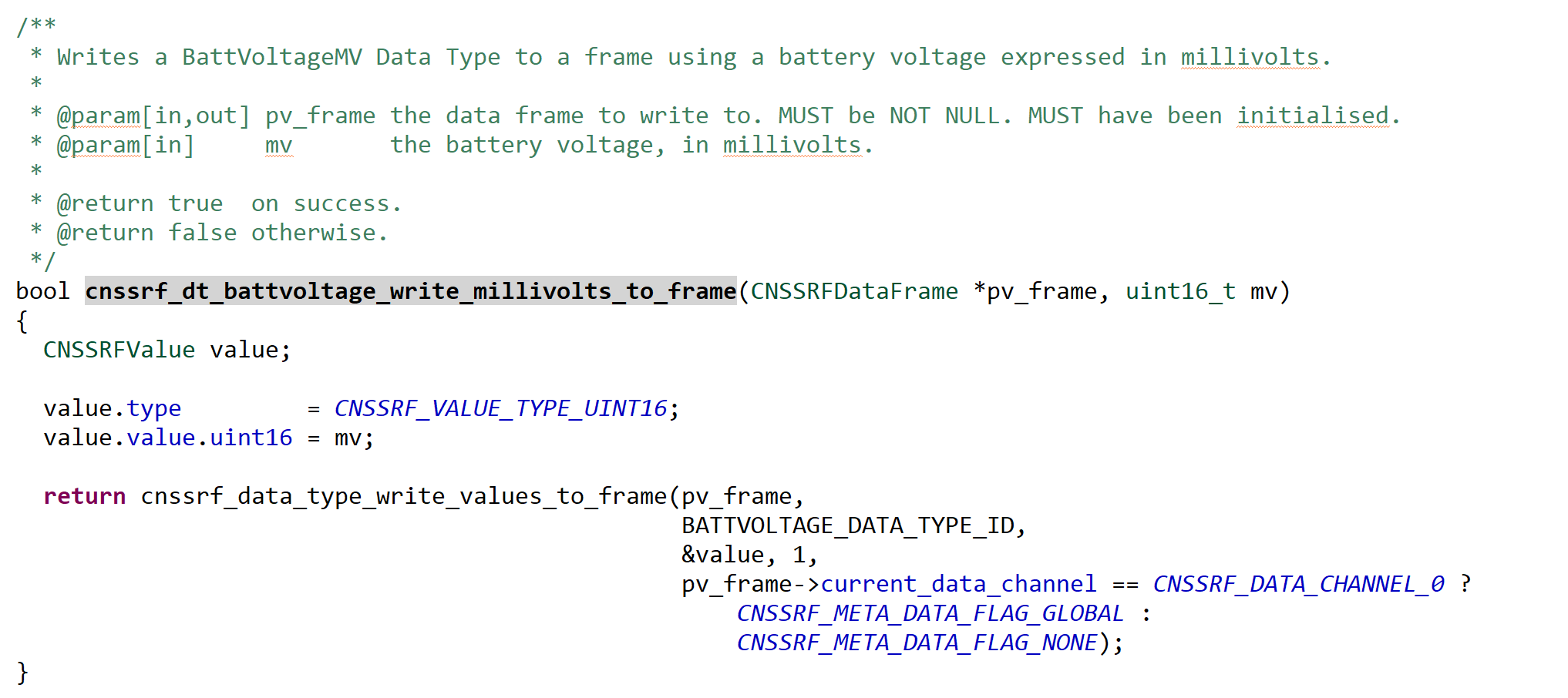
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Octets** | | | | | | | | | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ValueLSB | | | | | | | | Réservé | Réservé | Réservé | LowBatt | ValueMSB | | | |



Ou



(uint16)(volts\*1000)



La suite est équivalente



Exemple 15.34V \*1000 (uint16) : 0011 1011 1110 1100

Décodage

Payload[1] 0011 1011 octet1

Payload[0] 1110 1100 octet0

Value = 0011 1011

1110 1100

De 0 à 65,536V au millivolt près entier non signé de 16bits

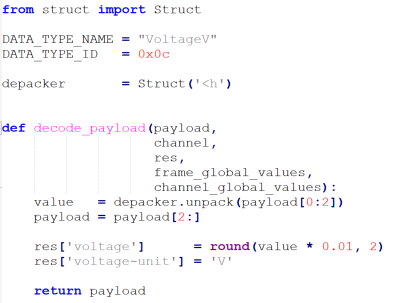
Pour VoltageC (0x0C)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Octets** | | | | | | | | | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ValueLSB | | | | | | | | ValueMSB | ValueMSB | ValueMSB | ValueMSB | ValueMSB | | | |

De -327,68 à +327,67V (int16) de -32768 to +32767 et uint\_16 = 65535

Donc 15.34V \*100 = 1534 (int16) : 0000 0101 1111 1110

Décodage



Struct.unpack(payload[0 :2]) = value [0 :2] à partir la liste indice 0 je prends 2 valeurs [1,9,7,7,5,3] ->[1,9]

Payload[2 :] unit = [2 :0] j’affiche tout ce qui est après l’indice 2 [1,9,7,7,5,3] ->[7,7,5,3]

Struct interprète les octets comme des données binaires compressées

Ce module effectue des conversions entre les valeurs Python et les structures C représentées sous forme d' [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) objets Python.

Récap : 0011 0001 0011 0000 = 12 592 : 11 0001 0011 0000

from struct import Struct

payload = b'011000100110000' '11 0001 0011 0000'=125 593 = 11 0001 0011 0001

depacker       = Struct('<h')

value   = depacker.unpack(payload[0:2]) #125 592

print(value)

[0000 0101] [1111 1110] j’ai l’impression qu’il y a qu’un payload

Octet1MSB Octet 0 LSB

Payload[1] 0000 0101 octet1

Payload[0] 1111 1110octet0

packed = struct.pack('i 4s f', 10, b'John', 2500)

print(packed) b'\n\x00\x00\x00John\x00@\x1cE'

packed = b'\n\x00\x00\x00John\x00@\x1cE'

unpacked = struct.unpack('i 4s f', packed)

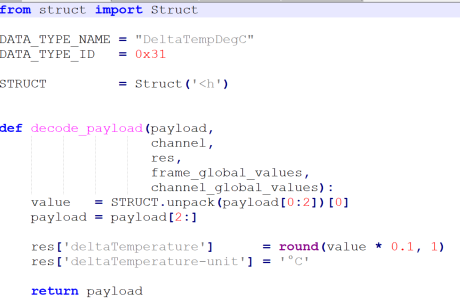
print(unpacked) (10, b'John', 2500.0)

Ce qui rapproche le DeltaTempDegC (0x31)

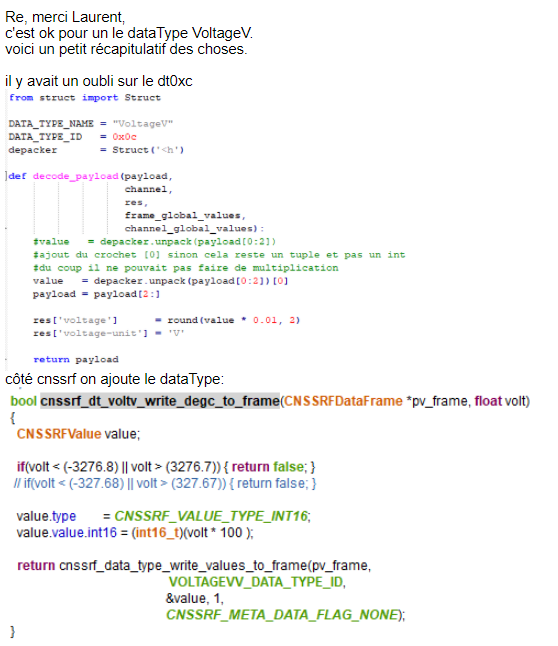
La valeur est : Octet\_0 | (Octet\_1 << 8).

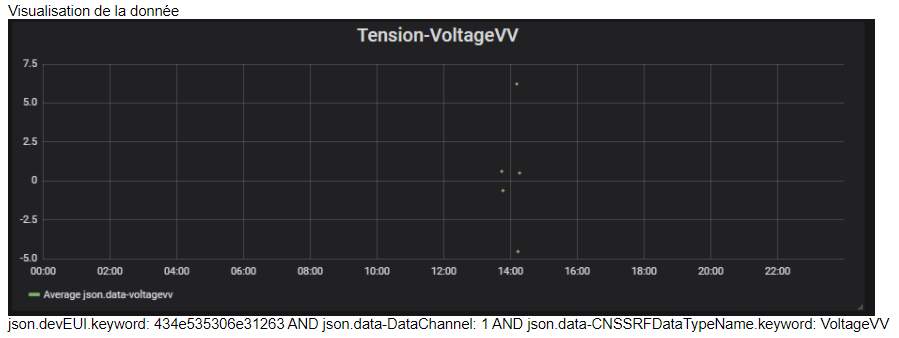
Le deltaT est : valeur x 0,1 °C. sur entier signé de -3 276.7 à +3276.7 le – erreur

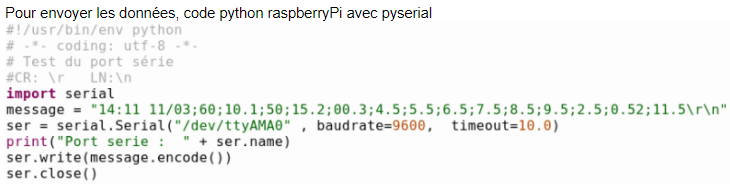
value.value.int16 = (int16\_t)(degc \* 10 + 0.5);



38-25= 13 -> (13\*10+0.5)=130.5 en int16\_t ( 1000 0010)

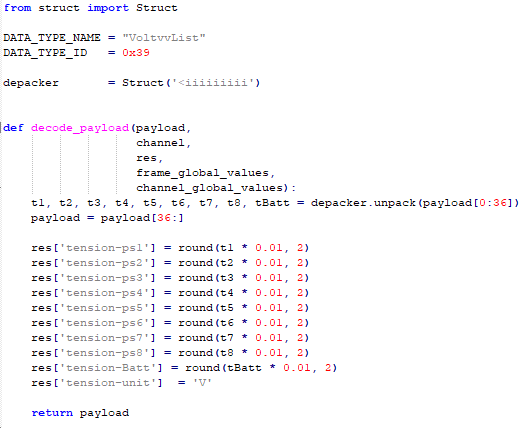


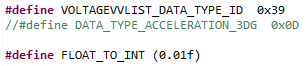


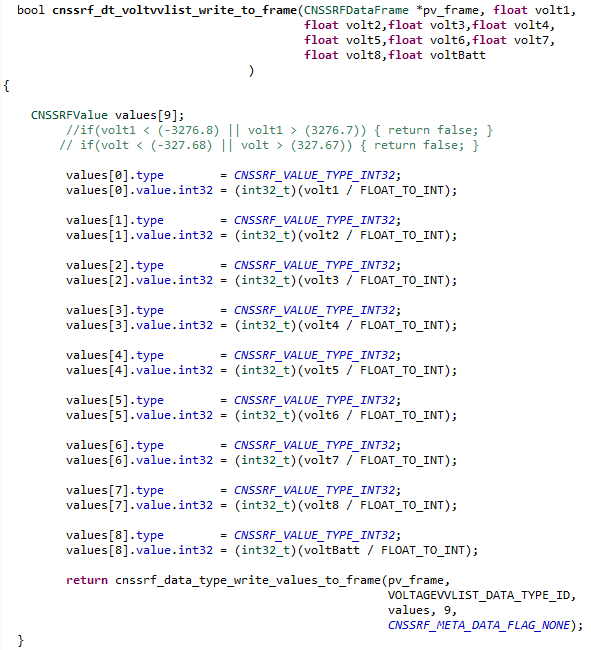


# **7 Conception de la Trame à envoyer et décoder (Tableau)**

|  |  |
| --- | --- |
| **VoltagevvList** | |
| Envoyer 8 tensions ps et 1 tension batterie (T1 ;T2 ;T3 ;T4 ;T5 ;T6 ;T7 ;T8 ;TBatt). Chaque composante est codée au moyen d’un entier signé sur 16 bits où chaque bit correspond une tension. | |
| DataTypeId : | 0x39 |
| Portée globale : | Non |
| Taille *Data Type* : | 1 octet |
| Taille des données : | octets : T1 (int32\_t LSB), T2 (int32\_t LSB), T3 (int32\_t LSB), T4 (int32\_t LSB), T5 (int32\_t LSB), T6 (int32\_t LSB), T7 (int32\_t LSB), T8 (int32\_t LSB), TBatt (int32\_t LSB), |







# 7 Liens

<https://github.com/myDevicesIoT/cayenne-docs/blob/master/docs/LORA.md>

<https://www.thethingsnetwork.org/docs/devices/arduino/api/cayennelpp/>

<https://python.sdv.univ-paris-diderot.fr/07_fichiers/>