CHECKSUM DES TRAMES DE TÉLÉINFORMATION ENEDIS

REV 2024-04-28

[!info] Voir la page [[Suivi de la téléinformation ERDF avec arduino]] # Documentation Enedis

ENEDIS L'ELECTRICITE EN RESEAU

Sorties de télé-information client des appareils de comptage Linky utilisés en généralisation par Enedis

Table 6 - Information systématiquement horodatée par le compteur

Format d'un groupe contenant une donnée horodatée							
< LF > (0x0 A)	Etiquette	< HT > (0x09)	Horodate	< HT > (0x09)	Donnée	< HT > (0x09)	Checksum	< CR > (0x0D)
	Zone contrôlée par la checksum							

Table 7 - Information sans horodate

Format d'un groupe contenant une donnée non horodatée						
< LF > (0x0 A)	Etiquette	< HT > (0x09)	Donnée	< HT > (0x09)	Checksum	< CR > (0x0D)
	Zone contrôlée par la checksum					

L'ordre d'émission des données dans la trame est celui donné par la lecture de haut en bas des tableaux donnés dans le chapitre 6.

La **checksum** est calculée sur l'ensemble des caractères allant du début du champ **Etiquette** à la fin du champ Donnée, caractères < HT > inclus.

Le principe de calcul de la **Checksum** est le suivant :

- calcul de la somme « S1 » de tous les caractères allant du début du champ « **Etiquette** » jusqu'au délimiteur (inclus) entre les champs « **Donnée** » et « **Checksum** ») ;
- cette somme déduite est tronquée sur 6 bits (cette opération est faite à l'aide d'un ET logique avec 0x3F) ;
- pour obtenir le résultat **checksum**, on additionne le résultat précédent S2 à 0x20.

En résumé :

Checksum = (S1 & 0x3F) + 0x20

Le résultat sera toujours un caractère ASCII imprimable compris entre 0x20 et 0x5F.

Python

```
def checksum(frame):
    somme = 0
    for c in frame[:-1]:
        somme += ord(c)
    somme_trunc = bin(somme)[-6:]
    calc_check = chr(32 + int(somme_trunc, 2))
    if calc_check == frame[-1:]:
```

```
        return True
    else:
        return False
```

1 JavaScript

```
function checksum(frame) {
    let somme = 0;
    for (let i = 0; i < frame.length - 1; i++) {
        somme += frame.charCodeAt(i);
    }
    const sommeTrunc = (somme & 0b111111).toString(2);
    const calcCheck = String.fromCharCode(32 + parseInt(sommeTrunc, 2));
    if (calcCheck === frame.slice(-1)) {
        return true;
    } else {
        return false;
    }
}
```

2 C

```
#include <stdbool.h>
#include <stddef.h>
#include <string.h>

bool checksum(const char *frame) {
    size_t longueur = strlen(frame);
    unsigned int somme = 0;
    if (longueur == 0) {
        return false;
    }

    for (size_t i = 0; i < longueur - 1; i++) {
        somme += (unsigned char)frame[i];
    }

    char calc_check = (char)(32 + (somme & 0b111111));
    return calc_check == frame[longueur - 1];
}
```

3 Rust

```rust
fn checksum(frame: &str) -> bool {
    let octets = frame.as_bytes();
    let mut somme: u32 = 0;
    if octets.is_empty() {
        return false;
    }

    for octet in &octets[..octets.len() - 1] {
        somme += u32::from(*octet);
    }

    let calc_check = 32 + (somme & 0b111111) as u8;
    calc_check == octets[octets.len() - 1]
}
```