Protocole EnOcean

Sommaire :

I - Présentation *1*
I.1. *Introduction* 1
I.2. *ESP – EnOcean Serial Protocol* 1
I.3. *EEP – EnOcean Equipment Profile* 4
II - Traitement d'une trame EnOcean *5*
II.1. *Algorithme principal* 5
II.2. *Modèle des classes* 5
II.3. *Éléments de codage* 6

I - Présentation

I.1. Introduction

La **documentation** complète du protocole **EnOcean** est disponible à l'adresse : https://www.enocean.com/fileadmin/redaktion/pdf/tec_docs/EnOceanSerialProtocol3.pdf.

EnOcean est une technologie **radio sans pile** brevetée. Elle est principalement déployée dans les installations domotiques du bâtiment.

A partir d'un mouvement linéaire, de la lumière ou d'une différence de température, **l'énergie récupérée permet l'émission** d'un signal radio capable par exemple d'allumer une lumière, d'émettre un changement d'état ou encore de modifier la température d'un radiateur.

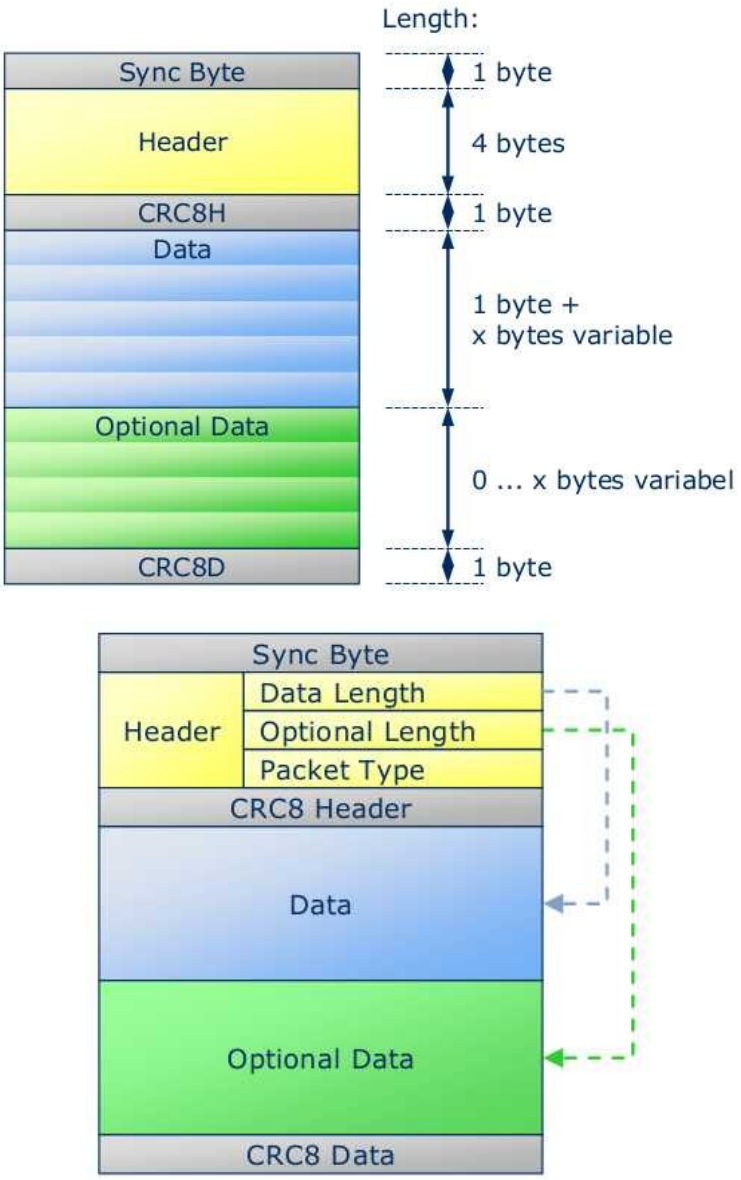
EnOcean couvre aujourd'hui un grand nombre de possibilités avec des modules qui utilisent une fréquence radio de **868,3 MHz** capables d'émettre sur une distance de **30 mètres** en intérieur. Les modules sont dotés de convertisseurs qui assurent la conversion entre le **signal radio** et une **liaison série**. Chaque module **EnOcean** est identifié avec une **adresse** de **4 octets unique**.

I.2. ESP – EnOcean Serial Protocol

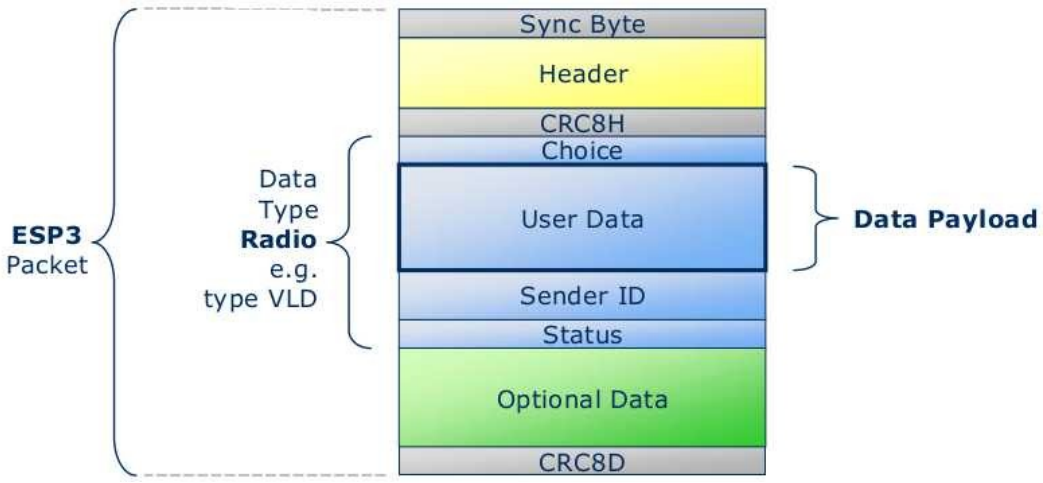
ESP (**E**nOcean **S**erial **P**rotocol) est le **protocole ouvert** de **communication série** entre un hôte et un module **EnOcean** (cf. spécifications : EnOcean Serial Protocol 3 (ESP3) V1.37 / August 10, 2017).

La **liaison série** est configurée de la façon suivante : **57600 bauds**, **8 bits de données**, **pas de bit de parité**, un bit de start (logical 0) et un bit de stop (logical 1).

La **trame** est composée d'une **entête**, d'un **champ de données**, d'un **champ optionnel** et d'un **champ de contrôle CRC8** dans sa version 3 (**ESP3**).



Un **message radio** (**RADIO_ERP1**) est encapsulé dans un paquet **ESP3** comme illustré ci-dessous :



Dans un fonctionnement de type message radio (**RADIO_ERP1**), le dialogue est très simple :

- Question suivi d'une réponse du module concerné ;
- ou Envoi d'un événement par un module.

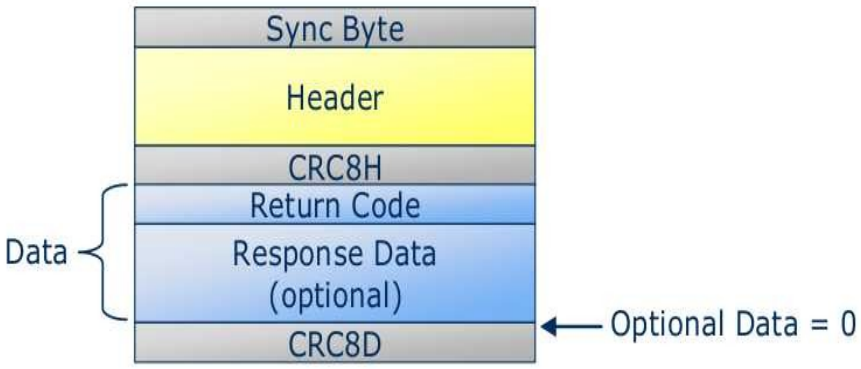
Si le temps entre la question et la réponse excède **500 ms** on considérera une erreur de transmission (**ESP3 Timeout**).

La **question** a la structure suivante :

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of radio telegram
	3	1	Optional Length	0x07	7 fields fixed
	4	1	Packet Type	0x01	RADIO_ERP1 = 1
-	5	1	CRC8H	0xnn	
Data	6	x	... ...	... ...	Radio telegram without checksum/CRC x = variable length / size
Optional Data	6+x	1	SubTelNum	0xnn	Number of subtelegram; Send: 3 / receive: 1 ... y
	7+x	4	Destination ID	0xnnnnnnnn	Broadcast radio: FF FF FF FF ADT radio: Destination ID (= address)
	11+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received subtelegrams (value decimal without minus)
	12+x	1	SecurityLevel	0x0n	0 = telegram unencrypted n = type of encryption (not supported any more)
-	13+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

La principale information utile du champ de données optionnelles sera le **RSSI** (Received Signal Strength Indication en db).

La **réponse** a la structure suivante (**Header** identique à la question) :



Le champ de contrôle **CRC8** est généré à l'aide du polynôme : **$G(x) = x^8 + x^2 + x^1 + x^0$**.

I.3. EEP – EnOcean Equipment Profile

Documentation des **EEP** :
https://www.enocean-alliance.org/wp-content/uploads/2020/07/EnOcean-Equipment-Profiles-3-1.pdf

EEP (**E**nOocean **E**quipment **P**rofile) ou **profil EEP** (codé sur 3 octets) défini le **type** des champs de **données** et d'**options** de chaque module :

RORG : Le premier octet indique le type de message radio ;
FUNC : Le deuxième octet indique la fonction de base de l'appareil. Détermine si l'appareil est un détecteur, un capteur, une sonde, un interrupteur, etc ;
TYPE : Le troisième octet indique les caractéristiques précises propres à l'appareil.

Seul l'octet de **RORG** est transmis dans une trame **ESP**. Les principaux **RORG** utilisés sont :

Telegram	RORG	Description
RPS	F6	Repeated Switch Communication
1BS	D5	1 Byte Communication
4BS	A5	4 Byte Communication
VLD	D2	Variable Length Data
MSC	D1	Manufacturer Specific Communication
ADT	A6	Addressing Destination Telegram
SM_LRN_REQ	C6	SMART ACK Learn Request
SM_LRN_ANS	C7	SMART ACK Learn Answer
SM_REC	A7	SMART ACK Reclaim
SYS_EX	C5	Remote Management
SEC	30	Secure telegram
SEC_ENCAPS	31	Secure telegram with RORG encapsulation
SEC_MAN	34	Maintenance Security message
SIGNAL	D0	Signal telegram
UTE	D4	Universal Teach In

Par exemple : **EEP = A5 02 05** :

RORG	A5	**4BS Telegram**
FUNC	02	Temperature Sensors
TYPE	05	Temperature Sensor Range 0°C to +40°C

Offset	Size	Bitrange	Data	ShortCut	Description	Valid Range	Scale	Unit
0	16	DB3.7...DB2.0	Not Used (= 0)					
16	8	DB1.7...DB1.0	Temperature	TMP	Temperature (linear)	255...0	0...+40	°C
24	4	DB0.7...DB0.4	Not Used (= 0)					
28	1	DB0.3	LRN Bit	LRNB	LRN Bit	Enum: 0: Teach-in telegram 1: Data telegram		
29	3	DB0.2...DB0.0	Not Used (= 0)					

A5 : **RORG**. Cette première valeur indique le type de message radio. Ici **A5** indique que le message radio émis par l'appareil est sur 4 octets ;

02 : **FUNC**. La fonction de base de l'appareil. Détermine si l'appareil est un détecteur, un capteur, une sonde, un interrupteur, etc. Ici **02** signifie que c'est une sonde de température ;

05 : **TYPE**. Caractéristiques précises propres à l'appareil. Par exemple pour ce capteur de température, **05** signifie que les valeurs possibles sont de 0°C à 40°C.

II - Traitement d'une trame EnOcean

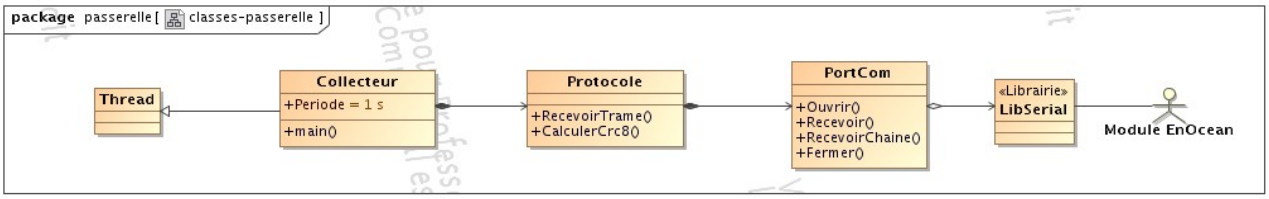
II.1. Algorithme principal

L'**algorithme** du programme principal de traitement d'une trame est le suivant :

```
Début
|       Faire Toujours
|       |       Attente Réception Trame
|       |       Si Trame reçue
|       |       |       Calcul du CRC8
|       |       |       Si CRC8 correct        // Trame valide
|       |       |       |       Récupération du type de trame (RORG)
|       |       |       |       Si  RORG = 0xA5
|       |       |       |       |       Récupération de la température
|       |       |       |       Fin Si
|       |       |       |       Si  RORG = 0xD5
|       |       |       |       |       Récupération état contact
|       |       |       |       Fin Si
|       |       |       |       Si (RORG != 0xA5 && RORG != 0xD5)
|       |       |       |       |       Afficher « RORG Inconnu »
|       |       |       |       Fin Si
|       |       |       Fin Si
|       |       |       SiNon
|       |       |       |       Afficher «  CRC8 Incorrect »
|       |       |       Fin SiNon
|       |       Fin Si
|       Fin Faire
Fin
```

II.2. Modèle des classes

Afin de mettre en œuvre cet algorithme, on peut utiliser le modèle des classes suivant :



L'**algorithme principal** est implémenté dans la fonction **main()**. il est codé en langage C++ et la trame est déclarée dans le **main()** de la façon suivante :

unsigned char trame[100];

La trame est gérée par une classe **Protocole** comportant 2 méthodes :

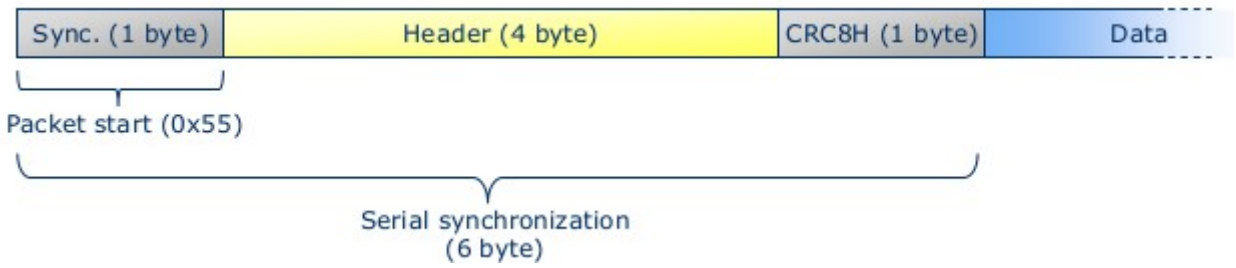
- **int Protocole::LireTrame(unsigned char *trame, int *taille)** : qui affecte au paramètre **trame** la trame reçue et la taille de celle-ci au paramètre **taille** ;
- **unsigned char Protocole::CalculerCrc8(unsigned char *trame, int taille)** : qui calcule et retourne le **crc8** de la **trame** passée en paramètre.

II.3. Éléments de codage

Le code de la méthode **RecevoirTrame()** suivant permet simuler la réception d'une trame :

```
int Protocole::RecevoirTrame(unsigned char *data)
{
unsigned char trame[] = {0x55, 0x00, 0x0a, 0x07, 0x01, 0xeb, 0xa5, 0x00, 0x00, 0x57,
0x08, 0x01, 0x81, 0x68, 0xe5, 0x00, 0x01, 0xff, 0xff, 0xff, 0xff, 0x37, 0x00, 0x23};
int taille = (trame[1]<<8) + trame[2] + trame[3] + 7;
for (int i=0; i<taille i++)
        {
        data[i] = trame[i];
        }
return 1 ;
}
```

Pour développer la méthode **RecevoirTrame()** il faut suivre les étapes suivantes :



- il faut attendre l'octet de synchronisation **0x55** ;
- il faut lire les 4 octets de l'entête et 1 octet du **CRC8H** ;
- il faut calculer le **CRC8H** des 4 octets de l'entête ;
- le comparer à l'octet de **CRC8H** lu ;
- Si les **CRC8H** sont égaux :
  - le paquet est valide et le reste de la trame doit être lue.
- Si les **CRC8H** sont différents, le **0x55** ne correspond pas à un octet de synchronisation et on doit recommencer le cycle de détection en attendant le prochain octet égal à **0x5**5.

Pour calculer le **CRC8** nous allons utiliser la fonction **uint8 proccrc8(uint8 crc8, unsigned char *trame)** définie dans le fichier **crc8.h** suivant :

```
// crc8.h

#ifndef _CRC8_H
#define _CRC8_H

#define uint8 unsigned char
```

```
uint8 u8CRC8Table[256] = {
0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15,
0x38, 0x3f, 0x36, 0x31, 0x24,0x23,0x2a,0x2d,
0x70, 0x77, 0x7e, 0x79, 0x6c,0x6b,0x62,0x65,
0x48, 0x4f, 0x46, 0x41, 0x54,0x53,0x5a,0x5d,
0xe0, 0xe7, 0xee, 0xe9, 0xfc,0xfb,0xf2,0xf5,
0xd8, 0xdf, 0xd6, 0xd1, 0xc4,0xc3,0xca,0xcd,
0x90, 0x97, 0x9e, 0x99, 0x8c,0x8b,0x82,0x85,
0xa8, 0xaf, 0xa6, 0xa1, 0xb4,0xb3,0xba,0xbd,
0xc7, 0xc0, 0xc9, 0xce, 0xdb,0xdc,0xd5,0xd2,
0xff, 0xf8, 0xf1, 0xf6, 0xe3,0xe4,0xed,0xea,
0xb7, 0xb0, 0xb9, 0xbe, 0xab,0xac,0xa5,0xa2,
0x8f, 0x88, 0x81, 0x86, 0x93,0x94,0x9d,0x9a,
0x27, 0x20, 0x29, 0x2e, 0x3b,0x3c,0x35,0x32,
0x1f, 0x18, 0x11, 0x16, 0x03,0x04,0x0d,0x0a,
0x57, 0x50, 0x59, 0x5e, 0x4b,0x4c,0x45,0x42,
0x6f, 0x68, 0x61, 0x66, 0x73,0x74,0x7d,0x7a,
0x89, 0x8e, 0x87, 0x80, 0x95,0x92,0x9b,0x9c,
0xb1, 0xb6, 0xbf, 0xb8, 0xad,0xaa,0xa3,0xa4,
0xf9, 0xfe, 0xf7, 0xf0, 0xe5,0xe2,0xeb,0xec,
0xc1, 0xc6, 0xcf, 0xc8, 0xdd,0xda,0xd3,0xd4,
0x69, 0x6e, 0x67, 0x60, 0x75,0x72,0x7b,0x7c,
0x51, 0x56, 0x5f, 0x58, 0x4d,0x4a,0x43,0x44,
0x19, 0x1e, 0x17, 0x10, 0x05,0x02,0x0b,0x0c,
0x21, 0x26, 0x2f, 0x28, 0x3d,0x3a,0x33,0x34,
0x4e, 0x49, 0x40, 0x47, 0x52,0x55,0x5c,0x5b,
0x76, 0x71, 0x78, 0x7f, 0x6A,0x6d,0x64,0x63,
0x3e, 0x39, 0x30, 0x37, 0x22,0x25,0x2c,0x2b,
0x06, 0x01, 0x08, 0x0f, 0x1a,0x1d,0x14,0x13,
0xae, 0xa9, 0xa0, 0xa7, 0xb2,0xb5,0xbc,0xbb,
0x96, 0x91, 0x98, 0x9f, 0x8a,0x8D,0x84,0x83,
0xde, 0xd9, 0xd0, 0xd7, 0xc2,0xc5,0xcc,0xcb,
0xe6, 0xe1, 0xe8, 0xef, 0xfa,0xfd,0xf4,0xf3
};

#define proccrc8(u8CRC, u8Data) (u8CRC8Table[u8CRC ^ u8Data])

#endif // _CRC8_H
```