Class MODBUS V1.0

Documents de référence :

Ces documents sont disponibles sur http://www.modbus.org
- *Modbus Application Protocol Specification*
- *Modbus over Serial Line – Specification and Implementation guide*

Historique des modifications

Version	Date	Nature et origine de la modification
1.0	07/10/05	Création du document.
1.0	11/10/05	Mise à jour – modification des méthodes avec changement des **char** en **byte**, mise à jour des algorithmes.
1.0	29/10/05	Modifications des algorithmes pour ne retourner qu'une trame de 2 octets en cas d'erreur sur un des paramètres passé. Optimisation.

1 Description :

La *class* **ModBus** implémente le protocol Modbus, qui définit un simple **PDU** (Protocol Data Unit) indépendant de la couche de communiation.

Cette *class* a donc en charge la construction des **PDU** pour les différentes fonctions Modbus suivantes :

- **Read Discret Input,**
- **Read Coils,**
- **Write Single Coil,**
- **Write Multiple Coils,**
- **Read Input Register,**
- **Read Holding Register,**
- **Write Single Register,**
- **Write Multiple Register,**
- **Read/Write Multiple Register,**
- **Read Device Identification**.

Elle aura aussi en charge :
- le traitement des erreurs Modbus,
- l'extraction des données contenues dans les trames de retour envoyées par le serveur Modbus.

2 Localisation :

Cette *class* sera localisée dans un package appelé : **schneider**

3 Prototype des méthodes de la *class*

◆ *public byte[]* ***readCoils****(int* ***st_add****, int* ***nb_coils****)* :
cette méthode construit une requête PDU permettant la lecture du status de 1 à 2000 bobines (sorties) successives dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse de la première bobine à lire.
Paramètre : ***int nb_coils,*** spécifie le nombre de bobines consécutives à lire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***nb_coils*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***readDiscreteInputs****(int* ***st_add****, int* ***nb_inputs****)* :
cette méthode construit une requête PDU permettant la lecture du status de 1 à 2000 entrées successives dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse de la première entrée à lire.
Paramètre : ***int nb_inputs,*** spécifie le nombre d'entrées consécutives à lire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***nb_inputs*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***readHoldingRegisters****(int* ***st_add****, int* ***nb_hregisters****)* :
cette méthode construit une requête PDU permettant la lecture du contenu de registres successifs dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse du premier registre à lire.
Paramètre : ***int nb_hregisters,*** spécifie le nombre de registres consécutifs à lire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***nb_hregisters*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***readInputRegisters****(int* ***st_add****, int* ***nb_iregisters****)* :
cette méthode construit une requête PDU permettant la lecture du contenu de 1 à approximativement 125 registres d'entrée successifs dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse du premier registre d'entrée à lire.
Paramètre : ***int nb_iregisters,*** spécifie le nombre de registres d'entrée consécutifs à lire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***nb_iregisters*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***writeSingleCoil****(int* ***add****, boolean* ***value****)* :
cette méthode construit une requête PDU permettant l'écriture d'une bobine (sortie) dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int add***, spécifie l'adresse de la bobine à écrire.
Paramètre : ***boolean value,*** spécifie la valeur à écrire dans la bobine (False pour **OFF** et True pour **ON**).
Retour : *char[],* tableau de caractères contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***add*** la fonction retourne **127**. Cette valeur peut être lue dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***writeSingleRegister****(int* ***add****, int* ***rvalue****)* :
cette méthode construit une requête PDU permettant l'écriture d'un registre dans un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int add***, spécifie l'adresse du registre à écrire.
Paramètre : ***int rvalue,*** spécifie la valeur à écrire dans le registre.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***add*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***rvalue*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***writeMultipleCoils****(int* ***st_add****, int* ***nb_coils,*** *byte[]* ***value****)* :
cette méthode construit une requête PDU permettant l'écriture d'une séquence de bobines (sorties) dans un équipement distant.

Elle prend trois paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse de la première bobine à écrire.
Paramètre : ***int nb_coils,*** spécifie le nombre de bobines à écrire.
Paramètre : ***byte[] value,*** tableau contenant les valeurs à écrire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127,** en cas d'erreur sur le paramètre ***nb_coils*** la fonction retourne **126** et en cas d'erreur sur la longueur du tableau ***value*** (value.length) la fonction retourne **125**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***writeMultipleRegisters****(int* ***st_add****, int* ***nb_registers,*** *byte[]* ***value****)* :
cette méthode construit une requête PDU permettant l'écriture d'une séquence de registres dans un équipement distant.

Elle prend trois paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_add***, spécifie l'adresse du premier registre à écrire.
Paramètre : ***int nb_registers,*** spécifie le nombre de registres à écrire.
Paramètre : ***byte[] value,*** tableau contenant les valeurs à écrire.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_add*** la fonction retourne **127,** en cas d'erreur sur le paramètre ***nb_registers*** la fonction retourne **126** et en cas d'erreur sur la longueur du tableau ***value*** (value.length) la fonction retourne **125**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***readWriteMultipleRegisters****(int* ***st_addRead****, int* ***nb_registersRead,***
int ***st_addWrite,*** *int* ***nb_registersWrite,*** *byte[]* ***value****)* :
cette méthode construit une requête PDU permettant l'écriture et la lecture d'une séquence de registres dans un équipement distant, et ce en une seule transaction **Modbus**.

Elle prend cinq paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***int st_addRead***, spécifie l'adresse du premier registre à lire.
Paramètre : ***int nb_registersRead,*** spécifie le nombre de registres à lire.
Paramètre : ***int st_addWrite***, spécifie l'adresse du premier registre à écrire.
Paramètre : ***int nb_registersWrite,*** spécifie le nombre de registres à écrire.
Paramètre : ***byte[] value,*** tableau contenant les valeurs à écrire dans les registres.
Retour : ***byte[]****,* tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***st_addRead*** la fonction retourne **127,** en cas d'erreur sur le paramètre ***st_addWrite*** la fonction retourne **126,** en cas d'erreur sur le paramètre ***nb_registersRead*** la fonction retourne **125**, en cas d'erreur sur le paramètre ***nb_registersWrite*** la fonction retourne **124** et en cas d'erreur sur la longueur du tableau ***value*** (value.length) la fonction retourne la valeur **123**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public byte[]* ***readDeviceIdentification****(byte* ***deviceIdCode****, char* ***objectId****)* :
cette méthode construit une requête PDU permettant la lecture de l'identification et d' informations relatives à la description physique et fonctionnelle d'un équipement distant.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***byte deviceIdCode***, spécifie le type de requête à effectuer.
Paramètre : ***char objectId,*** identifie le premier objet à obtenir.
Retour : ***byte[],*** tableau d'octets contenant la requête PDU.

NOTE : En cas d'erreur sur le paramètre ***deviceIdCode*** la fonction retourne **127** et en cas d'erreur sur le paramètre ***objectId*** la fonction retourne **126**. Ces valeurs peuvent être lues dans le premier champ du tableau d'octets retourné.

◆ *public boolean* ***checkIfError****(byte* ***functionCode****, byte* ***exceptionCode****)* :
cette methode à en charge l'identification des erreurs renvoyées par le serveur Modbus dans la trame PDU.

Elle prend deux paramètres en entrée et retourne une valeur en sortie.
Paramètre : ***byte functionCode***, code de la fonction contenu dans la trame de réponse renvoyée par le serveur.
Paramètre : ***byte exceptionCode***, code d'exception contenu dans la trame de réponse.
Retour : ***boolean,*** permet de renseigner si il y a eu erreur ou non. Si la fonction retourne **false** aucune erreur a été détectée, si la fonction retourne **true** le serveur Modbus a retourné une erreur.

NOTE : le code de la fonction renvoyé par le serveur se trouve dans le premier champ de la trame de réponse (trame PDU).
Le code d'exception quand à lui se trouve soit dans le deuxième, soit dans le troisième champ de la trame de réponse, cela dépend de la fonction qui a été demandée.

◆ *public string* ***getErrorType****()* :
cette méthode permet de recupérer le type d'erreur détectée lors d'une transaction Modbus.
Elle ne prend aucun paramètre en entrée mais retourne sous la forme d'une chaine de caractère le type d'erreur rencontré (ex : GATEWAY TARGET DEVICE FAILED TO RESPOND).

◆ *public byte[]* ***extractData****(byte[]* ***receive_tramePDU****)* :
cette méthode permet d'extraire les données reçues dans une trame PDU.

Elle prend un paramètre en entrée et retourne une valeur en sortie.
Paramètre : ***byte[253] receive_tramePDU***, trame PDU renvoyée par le servur Modbus.
Retour : ***char[]***, tableau de caractères contenant les données extraites.

4 Variables privées de la *class*

String **errorType** :
cette variable contient la description d'une erreur.

5 Algorithmes des méthodes de la *class*

- *public byte[]* ***readCoils****(int* ***st_add****, int* ***nb_coils****)* :

tramePDU : tableau d' octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si nb_coils < 1 ou nb_coils > 2000 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 5 octets

tramePDU[0] = 0x01 (code de la fonction)
tramePDU[1] = (st_add >> 8) **ET** 0xFF
tramePDU[2] = st_add **ET** 0xFF
tramePDU[3] = (nb_coils >> 8) **ET** 0xFF
tramePDU[4] = nb_coils **ET** 0xFF

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***readDiscreteInputs****(int* ***st_add****, int* ***nb_inputs****)* :

création de variables :
tramePDU : tableau d'octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si nb_inputs < 1 ou nb_inputs > 2000 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 5 octets

tramePDU[0] = 0x02 (code de la fonction)
tramePDU[1] = (st_add >> 8) **ET** 0xFF
tramePDU[2] = st_add **ET** 0xFF
tramePDU[3] = (nb_inputs >> 8) **ET** 0xFF
tramePDU[4] = nb_inputs **ET** 0xFF

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***readHoldingRegisters****(int* ***st_add****, int* ***nb_hregister****)* :

création de variables :
tramePDU : tableau d'octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si nb_hregisters < 1 ou nb_hregisters > 125 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 5 octets

tramePDU[0] = 0x03 (code de la fonction)
tramePDU[1] = (st_add >> 8) **ET** 0xFF
tramePDU[2] = st_add **ET** 0xFF
tramePDU[3] = (nb_hregister >> 8) **ET** 0xFF
tramePDU[4] = nb_hregister **ET** 0xFF

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***readInputRegisters****(int* ***st_add****, int* ***nb_iregister****)* :

création de variables :
tramePDU : tableau de d'octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si nb_iregisters < 1 ou nb_iregisters > 125 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 5 octets

tramePDU[0] = 0x04 (code de la fonction)
tramePDU[1] = (st_add >> 8) **ET** 0xFF
tramePDU[2] = st_add **ET** 0xFF
tramePDU[3] = (nb_iregister >> 8) **ET** 0xFF
tramePDU[4] = nb_iregister **ET** 0xFF

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***writeSingleCoil****(int* ***add****, boolean* ***value****)* :

création de variables :
tramePDU : tableau d'octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si add < 0 ou add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 *//remontée d'une erreur de paramètres*

sinon

création de variables :
coilValue : entier permettant de spécifier la valeur à écrire dans la bobine (sortie)
tramePDU : tableau de 5 octets

test des paramètres passés :
si value = FALSE **alors**
coilValue = 0x0000
sinon
coilValue = 0xFF00
fin si

construction de la trame
tramePDU[0] = 0x05 (code de la fonction)
tramePDU[1] = (add >> 8) **ET** 0xFF
tramePDU[2] = add **ET** 0xFF
tramePDU[3] = (coilValue >> 8) **ET** 0xFF
tramePDU[4] = coilValue **ET** 0xFF

fin si

une fois la trame construite, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***writeSingleRegister****(int* ***add****, int* ***rvalue****)* :

création de variables :
tramePDU : tableau d'octets

si add < 0 ou add > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si rvalue < 0 ou rvalue > 65535 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 5 octets

construction de la trame :
tramePDU[0] = 0x06 (code de la fonction)
tramePDU[1] = (add >> 8) **ET** 0xFF
tramePDU[2] = add **ET** 0xFF
tramePDU[3] = (rvalue>> 8) ET 0xFF
tramePDU[4] = rvalue **ET** 0xFF

fin si

une fois la trame construite, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***writeMultipleCoils****(int* ***st_add****, int* ***nb_coils,*** *byte[]* ***value****)* :

création de variables :
tramePDU :tableau d'octets
byteCount : entier contenant le nombre d'octets à écrire
rs : entier permettant d'enregistrer le reste de la division de nb_coils/8

calcul du byteCount
rs = nb_coils MOD 8
si rs = 0 **alors**
byteCount = nb_coils/8
sinon
//On ajoute 1 a la partie entière du résultat de la division. Pour calculer cette partie
//entière on applique le principe de la division euclidienne.
byteCount = ((nb_coils – rs)/8) + 1
fin si

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**
tramePDU : tableau de 2 octets
tramePDU[0] = 127 *//remontée d'une erreur de paramètres*
sinon si nb_coils < 1 ou nb_coils > 1968 **alors**
tramePDU : tableau de 2 octets
tramePDU[0] = 126 *//remontée d'une erreur de paramètres*
sinon si (value.longueur) > 246 **alors**
tramePDU : tableau de 2 octets
tramePDU[0] = 125 *//remontée d'une erreur de paramètres*
sinon si (value.longueur != byteCount) **alors**
tramePDU : tableau de 2 octets
tramePDU[0] = 124 *//remontée d'une erreur de paramètres*
sinon

création de variables :
lenghtTrame : entier permettant d'enregister la longueur de la trame
i : entier servant d'index à la boucle **for**

calcul de la longueur de la trame PDU
lengthTrame = 6 + byteCount

création de la variable tramePDU
tramePDU : tableau de *lengthTrame* octets

construction de la trame PDU :
tramePDU[0] = 0x0F (code de la fonction)
tramePDU[1] = (st_add >> 8) **ET** 0xFF
tramePDU[2] = st_add **ET** 0xFF
tramePDU[3] = (nb_coils >> 8) ET 0xFF
tramePDU[4] = nb_coils **ET** 0xFF
tramePDU[5] = (byte) byteCount

pour i = 0, i < byteCount, i++ **faire**
tramePDU[6+i] = value[i]
fin for

fin si
une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***writeMultipleRegisters****(int* ***st_add****, int* ***nb_registers,*** *byte[]* ***value****)* :

création de variables :
byteCount : entier contenant le nombre d'octets à écrire

calcul du byteCount
byteCount = nb_registers * 2

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_add < 0 ou st_add > 65535 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 127 //*remontée d'une erreur de paramètres*
sinon si nb_registers < 1 ou nb_registers > 120 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 126 //*remontée d'une erreur de paramètres*
sinon si (value.longueur) > 240 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 125 //*remontée d'une erreur de paramètres*
sinon si (value.longueur != byteCount) **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 124 //*remontée d'une erreur de paramètres*
sinon
    création de variables
    lenghtTrame : entier permettant d'enregister la longueur de la trame
    i : entier servant d'index à la boucle **for**

    calcul de la longueur de la trame PDU
    lengthTrame = 6 + byteCount

    création de la variable tramePDU
    tramePDU : tableau de *lengthTrame* octets

    construction de la trame PDU :
    tramePDU[0] = 0x10 (code de la fonction)
    tramePDU[1] = (st_add >> 8) **ET** 0xFF
    tramePDU[2] = st_add **ET** 0xFF
    tramePDU[3] = (nb_registers >> 8) **ET** 0xFF
    tramePDU[4] = nb_registers **ET** 0xFF
    tramePDU[5] = (byte) byteCount

    pour i = 0, i < byteCount, i++ **faire**
        tramePDU[6+i] = value[i]
    fin for

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***readWriteMultipleRegisters****(int* ***st_addRead****, int* ***nb_registersRead,*** *int* ***st_addWrite,*** *int* ***nb_registersWrite,*** *byte[]* ***value)*** :

création de variables :
byteCount : entier contenant le nombre d'octets à écrire
calcul du byteCount
byteCount = nb_registersWrite * 2

test des paramètres passés et remplissage de la trame en fonction du résultat :
si st_addRead < 0 ou st_addRead > 65535 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 127 //*remontée d'une erreur de paramètres*
sinon si st_addWrite < 0 ou st_addWrite > 65535 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 126 //*remontée d'une erreur de paramètres*
sinon si nb_registersRead < 1 ou nb_registersRead > 118 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 125 //*remontée d'une erreur de paramètres*
sinon si nb_registersWrite < 1 ou nb_registersWrite > 118 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 124 //*remontée d'une erreur de paramètres*
sinon si (value.longueur) > 236 **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 123 //*remontée d'une erreur de paramètres*
sinon si (value.length != byteCount) **alors**
    tramePDU : tableau de 2 octets
    tramePDU[0] = 122 //*remontée d'une erreur de paramètres*
sinon

    création de variables :
    lenghtTrame : entier permettant d'enregister la longueur de la trame
    i : entier servant d'index à la boucle **for**
    calcul de la longueur de la trame PDU
    lengthTrame = 10 + byteCount
    création de la variable tramePDU
    tramePDU : tableau de *lengthTrame* octets

    construction de la trame PDU :
    tramePDU[0] = 0x17 (code de la fonction)
    tramePDU[1] = (st_addRead >> 8) **ET** 0xFF
    tramePDU[2] = st_addRead **ET** 0xFF
    tramePDU[3] = (nb_registersRead >> 8) ET 0xFF
    tramePDU[4] = nb_registersRead **ET** 0xFF
    tramePDU[5] = (st_addWrite >> 8) **ET** 0xFF
    tramePDU[6] = st_addWrite **ET** 0xFF
    tramePDU[7] = (nb_registersWrite >> 8) **ET** 0xFF
    tramePDU[8] = nb_registersWrite **ET** 0xFF
    tramePDU[9] = (byte) byteCount

    pour i = 0, i < byteCount, i++ **faire**
        tramePDU[10+i] = value[i]
    fin for

fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public byte[]* ***readDeviceIdentification****(byte* ***deviceIdCode****, char* ***objectId****)* :

création de variables :
tramePDU : tableau d'octets

test des paramètres passés et remplissage de la trame en fonction du résultat :
si deviceIdCode != 01 **et** deviceIdCode != 02
et deviceIdCode != 03 **et** deviceIdCode != 04 **alors**

tramePDU : tableau de 2 octets
tramePDU[0] = 127 //*remontée d'une erreur de paramètres*

sinon si objectId < 0 ou objectId > 255

tramePDU : tableau de 2 octets
tramePDU[0] = 126 //*remontée d'une erreur de paramètres*

sinon

tramePDU : tableau de 4 octets

construction de la trame PDU
tramePDU[0] = 0x2B (code de la fonction)
tramePDU[1] = 0x0E (MODBUS Encapsulated Interface)
tramePDU[2] = deviceIdCode
tramePDU[3] = (byte)objectId
fin si

une fois la trame remplie, la méthode la retourne :
return(tramePDU)

- *public string* ***getErrorType****()* :

recupération et retour de la variable errorType
return(errorType)

- *public boolean **checkError**(byte **functionCode**, byte **exceptionCode**)* :

création de variables :
resultCheck : booléen qui retourne la valeur du résultat de la vérification des erreurs

test des paramètres passés et affectation de resultCheck en fonction du résultat :
si functionCode = 0x80 ou functioCode = 0x82 ou functionCode = 0x83
ou functionCode = 0x84 ou functionCode = 0x85 ou functionCode = 0x86
ou functionCode = 0x87 ou functionCode = 0x8B ou functionCode = 0x8C
ou functionCode = 0x8F ou functionCode = 0x90 ou functionCode = 0x91
ou functionCode = 0x94 ou functionCode = 0x95 ou functionCode = 0x96
ou functionCode = 0x97 ou functionCode = 0x98 ou functionCode = 0xAB **alors**

si exceptionCode = 01 **alors**
errorType = ERROR:ILLEGAL FUNCTION
resultCheck = **true**
fin si

si exceptionCode = 02 **alors**
errorType = ERROR:ILLEGAL DATA ADDRESS
resultCheck = **true**
fin si

si exceptionCode = 03 **alors**
errorType = ERROR:ILLEGAL DATA VALUE
resultCheck = **true**
fin si

si exceptionCode = 04 **alors**
errorType = ERROR:SLAVE DEVICE FAILURE
resultCheck = **true**
fin si

si exceptionCode = 05 **alors**
errorType = ACKNOWLEDGE
resultCheck = **true**
fin si

si exceptionCode = 06 **alors**
errorType = ERROR:SLAVE DEVICE BUSY
resultCheck = **true**
fin si

si exceptionCode = 08 **alors**
errorType = ERROR:MEMORY PARITY ERROR
resultCheck = **true**
fin si

si exceptionCode = 0A **alors**
errorType = ERROR:GATEWAY PATH UNAVAILABLE
resultCheck = **true**
fin si

si exceptionCode = 0B **alors**

errorType = ERROR:GATEWAY TARGET DEVICE FAILED TO RESPOND
resultCheck = **true**
fin si

sinon
errorType = NO ERROR
resultCheck = **false**

fin si

la méthode retourne le résultat de la vérification des erreurs
return(resultCheck)

NOTE : errorType est une variable de type String, privée (voir plus haut).

- *public byte[]* ***extractData****(byte[253]* ***receive_tramePDU****)* :

création des variables :
byteCount : octet contenant la valeur du nombre d'octets de données
fifoCount : entier contenant le nombre d'octets dans lus dans la FIFO pour la fonction 0x18
functionCode : octets contenant le code de la fonction exécutée par le serveur Modbus
dataArray : tableau d'octets
i : short integer servant d'index dans la boucle **for**

initialisation de variables
functionCode = receive_tramePDU[0]

//le traitement des données se fait par rapport au code de la fonction exécutée par le serveur Modbus
récupération des données pour les fonctions 0x01, 0x02, 0x03, 0x04 et 0x17
si functionCode = 0x01 ou functionCode = 0x02 ou functionCode = 0x03
ou functionCode = 0x04 ou functionCode = 0x17 **alors**

byteCount = receive_tramePDU[1]

intialisation du tableau de données
dataArray = tableau de byteCount octets

remplissage du tableau
pour i = 0; i<byteCount; i++ **faire**
dataArray[i] = receive_tramePDU[2+i]
fin for
fin si
récupération des données pour les fonctions 0x05, 0x06, 0x0F, 0x10 et 0x0B
//dans le cas de de ces deux fonctions la trame de retour contient un echo de ce que l'on a demané au serveur
si functionCode = 0x05 ou functionCode = 0x06 ou functionCode = 0x0F
ou functionCode = 0x10 ou functionCode = 0x0B **alors**

intialisation du tableau de données
dataArray = tableau de 4 octets

remplissage du tableau
pour i = 0; i<4; i++ **faire**
dataArray[i] = receive_tramePDU[1+i]
fin for

fin si

récupération des données pour la fonction 0x07
si functionCode = 0x07 **alors**

intialisation du tableau de données
dataArray = tableau de 1 octets

remplissage du tableau
dataArray[0] = receive_tramePDU[1]

fin si

récupération des données pour la fonction 0x08
si functionCode = 0x08 **alors**
(à faire)
fin si

récupération des données pour la fonction 0x0C
si functionCode = 0x0C **alors**

byteCount = receive_tramePDU[1]

intialisation du tableau de données
dataArray = tableau de byteCount octets

remplissage du tableau
pour i = 0; i< 6+ (byteCount – 6); i++ **faire**
dataArray[i] = receive_tramePDU[2+i]
fin for

fin si

récupération des données pour la fonction 0x11
si functionCode = 0x11 **alors**
(à faire)
fin si

récupération des données pour la fonction 0x14
si functionCode = 0x11 **alors**
(à faire)
fin si

récupération des données pour la fonction 0x15
si functionCode = 0x11 **alors**
(à faire)
fin si

récupération des données pour la fonction 0x16
si functionCode = 0x16 **alors**

intialisation du tableau de données
dataArray = tableau de 6 octets

remplissage du tableau
pour i = 0; i<6; i++ **faire**
dataArray[i] = receive_tramePDU[1+i]
fin for

fin si

récupération des données pour la fonction 0x18
si functionCode = 0x18 **alors**

création de la variable intermediateValue
int intermediateValue

fifoCount = (int)receive_tramePDU[2]
fifoCount = fifoCount << 8
intermediateValue = (int)reveive_tramePDU[3]
fifoCount = fifoCount OU intermediateValue

intialisation du tableau de données
dataArray = tableau de fifoCount octets

remplissage du tableau
pour i = 0; i< fifoCount; i++ **faire**
dataArray[i] = receive_tramePDU[5+i]
fin for

fin si

récupération des données pour la fonction 0x2B
si functionCode = 0x2B **alors**
(à faire)
fin si

la méthode retourne la tableau contenant les données
return(dataArray)