Example コードのロジック、使用データ、CAN 通信ファンクションブロック診断コード

内容

1. Example コード(Function Block ExampleMachine)の処理の流れ 3

1.1 E_1000（iExample = 1000） 1 つの Function Block Instance AXL_CAN_COMM_X1 で CAN メッセージ送受信 3

1.2 E_2000（iExample = 2000） 2 つの Function Block Instance AXL_CAN_COMM_X1(2)で CAN メッセージ送受信 6

1.3 E_3000（iExample = 3000） Function Block Instance AXL_CAN_Para で CAN マスタ AXL F IF CAN のパラメータ読み出し/設定 8

1.4 E_4000（iExample = 4000） Function Block Instance AXL_CAN_Para11 で CAN マスタ AXL F IF CAN の 11bit/29bit フィルタ設定.. 10

1.5 FINISH（iExample = 32000） 終了処理 11

2. Example コードで使用するデータ構造 12

2.1 EXA_UDT_EXAMPLE ： STRUCT ExampleMachine で使用される udtExample のデータ構造 12

2.2 EXA_UDT_AXL_CAN_COMM ： STRUCT Function Block AXL_CAN_COMM 入出力引数 13

2.3 EXA_UDT_AXL_CAN_PARA ： STRUCT Function Block AXL_CAN_Para 入出力引数 15

2.4 EXA_UDT_AXL_CAN_PARA11 ： STRUCT Function Block AXL_CAN_Para11 入出力引数 17

2.5 EXA_UDT_AXL_CAN_PARA29 ： STRUCT Function Block AXL_CAN_Para29 入出力引数 18

2.6 EXA_UDT_ASYNCOM_AXL ： STRUCT Function Block AsynCom_AXL 入出力引数 19

2.7 AXL_CAN_UDT_DIAGSTATE ： STRUCT Function Block AXL_CAN_Param の出力引数 srDiagState の構造 19

2.8 ASYN_UDT_COM ： STRUCT Function Block AsynCom_AXL が使用するデータ構造 20

2.9 CAN_UDT_DATA ： STRUCT CAN の送受信データを入れるデータ構造 21

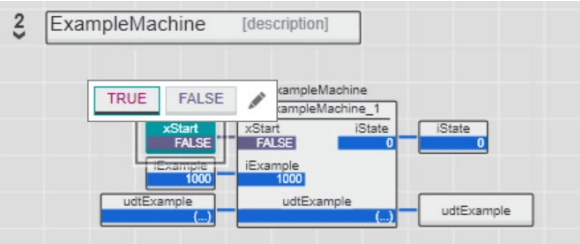
2.10 CAN_ARR_MESSAGES_0_199 ： ARRAY[0..199] OF CAN_UDT_MESSAGE; CAN メッセージ配列 21

2.11 CAN_ARR_MESSAGES_0_3 ： ARRAY[0..3] OF CAN_UDT_MESSAGE; 高優先度 CAN メッセージ配列 21

2.12 CAN_UDT_LAST_MESSAGES ： STRUCT　　最新の送受信 CAN メッセージ 21
2.13 CAN_UDT_MESSAGE ： STRUCT　　1 つの CAN メッセージを包含する構造体 22
2.14 CAN_ARR_B_1_8 ： ARRAY[1..8] OF BYTE;　CAN メッセージペイロード部分 22
2.15 CAN_ARR_11BitFilterRange ： ARRAY[0..59] OF CAN_UDT_11BitFilter; 11bit フィルタ配列 22
2.16 CAN_ARR_29BitFilterRange ： ARRAY[0..29] OF CAN_UDT_29BitFilter; 29bit フィルタ配列 22
2.17 CAN_UDT_11BitFilter ： STRUCT　　11bit フィルター範囲 22
2.18 CAN_UDT_29BitFilter ： STRUCT　　29bit フィルター範囲 22
3. CAN Function Block 診断コード 23
3.1 AXL_CAN_COMM Diagnostic table 23
3.2 AXL_CAN_Para Diagnostic table 25
3.3 AXL_CAN_Para11 Diagnostic table 32
3.4 AXL_CAN_Para29 Diagnostic table 41

1. Example コード(Function Block ExampleMachine)の処理の流れ

Main プログラムの Function Block Instance ExampleMachine の入力引数 iExample に実行したい処理をセットし(デフォルトは 1000)、
xStart を FALSE→TRUE とすると実行開始。



1.1 E_1000 (iExample = 1000)　　1 つの Function Block Instance AXL_CAN_COMM_X1 で CAN メッセージ送受信

CAN 送受信準備→CAN 送信→CAN 受信→CAN 送信→CAN 受信→終了
udtExample.udtCanData を通じて CAN の送受信データにアクセス
udtExample.udtAXL_CAN_COMM_X1 を通じてそれ以外の Function Block Instance AXL_CAN_COMM_X1 の入出力データにアクセス。

iState = 0:
　　AXL_CAN_COMM_X1 起動
　　iState を 10 にセット
iState = 10:
　　AXL_CAN_COMM_X1 アクティブでエラーがなければ、
　　送信、受信を有効に。同じ ID のメッセージは receive-array(udtExample.udtCanData.arrMessageReceive)に上書きする設定

(Receive-mode 1)
iState を 20 にセット。
iState = 20:
AXL_CAN_COMM_X1 アクティブでエラーがなければ、udtExample.udtCanData.arrMessageSend[0]に送信する CAN メッセージをセット。
udtExample.udtCanData.arrMessageSend[0].xUsed に TRUE をセットし送信指示。
iState を 30 にセット。
iState = 30:
AXL_CAN_COMM_X1 アクティブでエラーがなければ、
iState を 40 にセット。
iState = 40:
AXL_CAN_COMM_X1 アクティブでエラーがなく、udtExample.udtCanData.arrMessageReceive[0]の受信したデータが想定通りで、
udiSequence(受信頻度)が 1 回だけ
だったら
送信内容は変更せず（もちろん変更してもよい）
udtExample.udtCanData.arrMessageSend[0].xUsed に TRUE をセットし再度送信指示。
iState を 50 にセット。
iState = 50:
AXL_CAN_COMM_X1 アクティブでエラーがなく、総送信メッセージ(udtExample.udtAXL_CAN_COMM_X1.udiMessageSend)が 2 だったら
iState を 60 にセット。
iState = 60:
AXL_CAN_COMM_X1 アクティブでエラーがなく、総送信メッセージ(udtExample.udtAXL_CAN_COMM_X1.udiMessageSend)が 2 で、
受信メッセージ(udtExample.udtCanData.arrMessageReceive[0])が想定通りで、
udiSequence(受信頻度)が 2 回だけ

だったら、
送信メッセージ(udtExample.udtCanData.arrMessageSend[0])と受信メッセージ(udtExample.udtCanData.arrMessageReceive[0])は
空にして、AXL_CAN_COMM_X1 も不起動にしてその他入力引数も 0 ないし FALSE にする。
iState を 70 にセット

iState = 70:

AXL_CAN_COMM がインアクティブになったら iState を 1000 にセット

iState = 1000

udtExample.iExample に 32000 をセット
udtExample.iState に 0 をセット

1.2 E_2000（iExample = 2000）　2 つの Function Block Instance AXL_CAN_COMM_X1(2)で CAN メッセージ送受信

CAN 送受信準備→CAN 送信→CAN 受信→CAN 送信→CAN 受信→終了
udtExample.udtCanData を通じて CAN の送受信データにアクセス
udtExample.udtAXL_CAN_COMM_X1 、 udtExample.udtAXL_CAN_COMM_X2 を通じてそれ以外の Function Block Instance AXL_CAN_COMM_X1 、AXL_CAN_COMM_X2 の入出力データにアクセス。

iState = 0:
　　AXL_CAN_COMM_X1、AXL_CAN_COMM_X2 起動
　　iState を 10 にセット
iState = 10:
　　AXL_CAN_COMM_X1(2)アクティブでエラーがなければ、
　　AXL_CAN_COMM_X1 を受信用に、AXL_CAN_COMM_X2 を送信用にセット。
　　同じ ID のメッセージは receive-array(udtExample.udtCanData.arrMessageReceive)に上書きする設定（Receive-mode 1）
　　iState を 20 にセット。
iState = 20:
　　AXL_CAN_COMM_X1(2)アクティブでエラーがなければ、udtExample.udtCanData.arrMessageSend[0]に送信する CAN メッセージをセット。
　　udtExample.udtCanData.arrMessageSend[0].xUsed に TRUE をセットし送信指示。
　　iState を 30 にセット。
iState = 30:
　　AXL_CAN_COMM_X2 アクティブでエラー(送信エラー)がなければ、
　　iState を 40 にセット。
iState = 40:

AXL_CAN_COMM_X1 アクティブでエラー(受信エラー)がなく、udtExample.udtCanData.arrMessageReceive[0]の受信したデータが想定通りで
udiSequence(受信頻度)が 1 回だけ
だったら、送信内容は変更せず（もちろん変更してもよい）
udtExample.udtCanData.arrMessageSend[0].xUsed に TRUE をセットし送信指示。
iState を 50 にセット。

iState = 50:

AXL_CAN_COMM_X2 アクティブでエラー(送信エラー)がなく、総送信メッセージ(udtExample.udtAXL_CAN_COMM_X2.udiMessageSend)が
2 だったら
iState を 60 にセット。

iState = 60:

AXL_CAN_COMM_X1 アクティブでエラー(受信エラー)がなく、総送信メッセージ(udtExample.udtAXL_CAN_COMM_X1.udiMessageSend)が
0 で(AXL_CAN_COMM_X1 は受信専用なので)、
受信メッセージ(udtExample.udtCanData.arrMessageReceive[0])が想定通りで
udiSequence(受信頻度)が 2 回だったら
送信メッセージ(udtExample.udtCanData.arrMessageSend[0])と受信メッセージ(udtExample.udtCanData.arrMessageReceive[0])は
空にして、AXL_CAN_COMM_X1 と AXL_CAN_COMM_X2 も不起動にしてその他入力引数も 0 ないし FALSE にする。
iState を 70 にセット

iState = 70:

AXL_CAN_COMM がインアクティブになったら iState を 1000 にセット

iState = 1000

udtExample.iExample に 32000 をセット
udtExample.iState に 0 をセット

1.3 E_3000（iExample = 3000）　Function Block Instance AXL_CAN_Para で CAN マスタ AXL F IF CAN のパラメータ読み出し/設定

パラメータ通信 FB 準備→パラメータ読み出し→パラメータ設定→終了
udtExample.udtAXL_CAN_Para を通じて Function Block Instance AXL_CAN_Para の入出力データにアクセス。

iState = 0:
　　udtExample.udtAsyncComAXL.wSlot = WORD#16#0001;　　スロット番号(コントローラの右隣から数えた順番)
　　AXL_CAN_Para 起動
　　iState を 10 にセット
iState = 10:
　　AXL_CAN_Para アクティブでエラーがなければ、
　　ビットレート読み出しを有効にする(udtExample.udtAXL_CAN_Para.xReadBitRate を TRUE)
　　iState を 20 にセット。
iState = 20:
　　AXL_CAN_Para アクティブでエラーがなければ、読み出したビットレート(udtExample.udtAXL_CAN_Para.udiBitRate)を一時保存
　　ビットレート読み出しを無効にする(udtExample.udtAXL_CAN_Para.xReadBitRate を FALSE)
　　iState を 30 にセット。
iState = 30:
　　ロケーション(udtExample.udtAXL_CAN_Para.strSetLocation)に’ExampleLocation’をセットし、
　　ロケーション設定を有効に(udtExample.udtAXL_CAN_Param.xSetLocation を TRUE)
　　iState を 40 にセット。
iState = 40:
　　AXL_CAN_Para アクティブでエラーがなく、udtExample.udtAXL_CAN_Param.xDone が TRUE（設定が成功していたら）

設定した udtExample.udtAXL_CAN_Para.strLocation を一時保存
ロケーション設定を無効に(udtExample.udtAXL_CAN_Param.xSetLocation を FALSE)
iState を 50 にセット。
iState = 50:
AXL_CAN_Para を起動停止しする(udtExample.udtAXL_CAN_Para.xActivate を FALSE)
iState を 60 にセット。
iState = 60:
AXL_CAN_Param がインアクティブになったら iState を 1000 にセット（* ソースコードで AXL_CAN_COMM_X1 を調べているのは誤り*)
iState = 1000
udtExample.iExample に 32000 をセット
udtExample.iState に 0 をセット

1.4 E_4000（iExample = 4000）　Function Block Instance AXL_CAN_Para11 で CAN マスタ AXL F IF CAN の 11bit/29bit フィルタ設定

11bitID フィルタ設定→29bit ID フィルタ設定→終了
udtAXL_CAN_Para11、udtAXL_CAN_Para29 を通じて Function Block Instance AXL_CAN_Para11、AXL_CAN_Para29 の入出力データにアクセス。

iState = 0:
　　11bitID 用フィルタ適用の ID 範囲（ID 下限 udtExample.udtAXL_CAN_Para11.arrFilterBitRanges[0].uiFrom ～
　　ID 上限 udtExample.udtAXL_CAN_Para11.arrFilterBitRanges[0].uiTo)をセット
　　11bitID 用フィルタの動作モードを bFilter11BitMode を 3 に設定(フィルタ範囲をブロック)
　　udtExample.udtAXL_CAN_Param11.wSlot = WORD#16#0001;　　スロット番号(コントローラの右隣から数えた順番)
　　AXL_CAN_Para11 起動
　　iState を 10 にセット
iState = 10:
　　AXL_CAN_Para11 アクティブでエラーがなければ、
　　iState を 20 にセット。
iState = 20:
　　29bitID 用フィルタ適用の ID 範囲（ID 下限 udtExample.udtAXL_CAN_Para29.arrFilterBitRanges[0].uiFrom ～
　　ID 上限 udtExample.udtAXL_CAN_Para29.arrFilterBitRanges[0].uiTo)をセット
　　29bitID 用フィルタの動作モードを bFilter29BitMode を 3 に設定(フィルタ範囲をブロック)
　　udtExample.udtAXL_CAN_Param29.wSlot = WORD#16#0001;　　スロット番号(コントローラの右隣から数えた順番)
　　AXL_CAN_Para29 起動
　　iState を 30 にセット。
iState = 30:

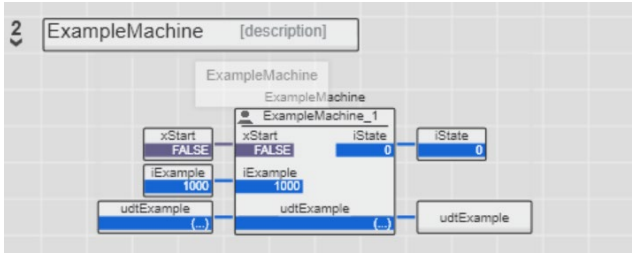
AXL_CAN_Para29 アクティブでエラーがなければ、
11bitID 用フィルタの設定をクリアし、AXL_CAN_Para11 を起動停止。
29bit 用フィルタの設定をクリアし、AXL_CAN_Para29 を起動停止。
iState を 40 にセット。

iState = 40:
AXL_CAN_Para11、AXL_CAN_Para29 ともインアクティブならば、iState に 1000 をセット

iState = 1000
udtExample.iExample に 32000 をセット
udtExample.iState に 0 をセット

1.5 FINISH（iExample = 32000）　終了処理

各 Example が最後まで終了すると、iExample 32000 に移動する。
Main program の Function Block Instance Example Machine の入力引数 xStart を FALSE にすると完全終了する。



2. Example コードで使用するデータ構造

udtExample : EXA_UDT_EXAMPLE;

2.1 EXA_UDT_EXAMPLE : STRUCT　　ExampleMachine で使用される udtExample のデータ構造

iExample	: INT;	処理の番号
iState	: INT;	処理毎のステート
(* Instances *)		
udtAXL_CAN_COMM_X1	: EXA_UDT_AXL_CAN_COMM;	Function Block Instance AXL_CAN_COMM_X1 入出力引数
udtAXL_CAN_COMM_X2	: EXA_UDT_AXL_CAN_COMM;	Function Block Instance AXL_CAN_COMM_X2 入出力引数
udtAXL_CAN_Para	: EXA_UDT_AXL_CAN_PARA;	Function Block Instance AXL_CAN_Para 入出力引数
udtAXL_CAN_Para11	: EXA_UDT_AXL_CAN_PARA11;	Function Block Instance AXL_CAN_Para11 入出力引数
udtAXL_CAN_Para29	: EXA_UDT_AXL_CAN_PARA29;	Function Block Instance AXL_CAN_Para29 入出力引数
udtAsynComAXL	: EXA_UDT_ASYNCOM_AXL;	Function Block Instance AsynCom_AXL 入出力引数
(* Exchange structures *)		
udtAsynCom	: ASYN_UDT_COM;	非同期通信のためのデータ構造。ユーザープログラムでの直接アクセス不要
udtCanData	: CAN_UDT_DATA;	送受信する CAN データを入れるデータ構造

END_STRUCT;

2.2 EXA_UDT_AXL_CAN_COMM : STRUCT

Function Block AXL_CAN_COMM 入出力引数
AXL_CAN_COMM: AXL F IF CAN 1H を通じて
CAN 通信を行う

```
(* 入力引数 *)
        xActivate               : BOOL;
        xReset                  : BOOL;
        tBusTimeout             : TIME;
        xSend                   : BOOL;
        xReceive                : BOOL;
        xReceiveMode            : BOOL;
        xPIExStopIn             : BOOL;
        xCANStopIn              : BOOL;
        xClearSendBuffer        : BOOL;
        xClearReceiveBuffer     : BOOL;
(* 出力引数 *)
        xActive                 : BOOL;
        xError                  : BOOL;
        wDiagCode               : WORD;
        wAddDiagCode            : WORD;
```

立ち上がりエッジで起動。FALSE で不起動。
立ち上がりエッジでリセット
メッセージ送信後の Ack 待ち時間（デフォルト 500 ms）
TRUE で send-array のメッセージをモジュール send_buffer に送信
TRUE でモジュールの receive-buffer から receive-array に受信
TRUE ： Receive-mode 1。同一 ID のメッセージは receive-array で
上書き。udiSequence に受信頻度を記載。
FALSE ： Receive-mode 2。受信メッセージは receive-array の次の空
フィールドに格納。
モジュール PI_Ex_Stop ビットを設定。TRUE で AXL F 側との通信を停止
モジュール CAN_Stop ビットを設定。TRUE で CAN バスとの通信を停止
TRUE: モジュールの send-buffer をクリア
TRUE: モジュールの receive-buffer をクリア

FALSE: インアクティブ
TRUE: アクティブ（起動後アクティブになるのを要確認）
TRUE: エラー発生。詳細は wDiagCode と wAddDiagCode に
診断コード。詳細は AXL_CAN_COMM Diagnostic table
追加診断コード。詳細は AXL_CAN_COMM Diagnostic table

```
        xPIExStopOut            : BOOL;             TRUE: CAN、モジュール間のデータ転送が停止
        xCANStopOut             : BOOL;             TRUE: CAN コントローラ停止中。
        xCAIN_OVR               : BOOL;             TRUE: receive-buffer オーバーラン。以後のメッセージは破棄
        xCC_OVR                 : BOOL;             TRUE: CAN コントローラ・オーバーラン。メッセージ消失
        xCC_WARN                : BOOL;             TRUE: CAN コントローラ警告レベルに到達
        xCC_BusOff              : BOOL;             TRUE: CAN コントローラバスが BusOff 状態。
        siSendBufferState       : SINT;             send-buffer の状態 0:空、1:0-33%、2:33-67%、3:67%以上
        siReceiveBufferState    : SINT;             receive-buffer の状態 0:空、1:0-33%、2:33-67%、3:67%以上
        usiSendBufferCount      : USINT;            モジュールの send-buffer にあるメッセージの数
        usiReceiveBufferCount   : USINT;            モジュールの receive-buffer にあるメッセージの数
        udiMessagesSend         : UDINT;            send-array からモジュールの send-buffer に送信されたメッセージの数
END_STRUCT;
```

2.3 EXA_UDT_AXL_CAN_PARA : STRUCT

Function Block AXL_CAN_Para 入出力引数
AXL_CAN_Para: AXL F IF CAN 1H のパラメータ
の読み書きを行う

```
(* 入力引数 *)
        xActivate               : BOOL;
        xReset                  : BOOL;
        strSetLocation          : STRING;
        xSetLocation            : BOOL;
        strSetEquipmentIdent    : STRING;
        xSetEquipmentIdent      : BOOL;
        xGetDiagState           : BOOL;
        tDiagStateCycle         : TIME;
        xResetDiag              : BOOL;
        xResetParam             : BOOL;
        udiSetBitRate           : UDINT;
        xSetBitRate             : BOOL;
        xReadBitRate            : BOOL;
(* 出力引数 *)
        xActive                 : BOOL;

        xError                  : BOOL;
        wDiagCode               : WORD;
        wAddDiagCode            : WORD;
```

立ち上がりエッジで起動。FALSE で不起動
立ち上がりエッジでリセット
ロケーション文字列
TRUE : ロケーションをセット
機器名称
TRUE : 機器名称をセット
TRUE : 診断状態を取得
診断状態読み出し周期（0 = 更新しない）
TRUE : 診断状態をリセット
TRUE : 全パラメータを初期化
ビットレート
TRUE : ビットレートをセット
TRUE : ビットレートを読み出し

FALSE: インアクティブ
TRUE: アクティブ（起動後アクティブになるのを要確認）
TRUE : エラー発生。詳細は wDiagCode、wAddDIagCode 参照
診断コード。詳細は AXL_CAN_Para Diagnostic table
追加診断コード。詳細は AXL_CAN_Para Diagnostic table

```
        xDone                   : BOOL;                 TRUE : コマンド成功（1 周期のみ）
        strFirmwareVersion      : STRING;               モジュールのファームウエアバージョン
        strLocation             : STRING;               モジュールのロケーション
        strEquipmentIdent       : STRING;               機器名称
        udtDiagState            : AXL_CAN_UDT_DIAGSTATE; モジュール診断情報を含む構造体
        udiBitRate              : UDINT;                現在のビットレート
END_STRUCT;
```

2.4 EXA_UDT_AXL_CAN_PARA11 : STRUCT　　Function Block AXL_CAN_Para11 入出力引数
AXL_CAN_Para11: AXL F IF CAN 1H の 11bit
フィルタの設定を行う

```
(* 入力引数 *)
        xActivate               : BOOL;         立ち上がりエッジで起動。FALSE で不起動
        xReset                  : BOOL;         立ち上がりエッジでリセット
        xResetParam             : BOOL;         TRUE : 全パラメータを初期化
        bFilter11BitMode        : BYTE;         11bit フィルタモード
                                                0: 全て受信, 1: 全てブロック,
                                                2: フィルタ有効(受信)、3: フィルタ有効(送信)
(* 出力引数 *)
        xActive                 : BOOL;         FALSE: インアクティブ
                                                TRUE: アクティブ（起動後アクティブになるのを要確認）

        xError                  : BOOL;         TRUE : エラー発生。詳細は wDiagCode、wAddDIagCode 参照
        wDiagCode               : WORD;         診断コード。詳細は AXL_CAN_COMM Diagnostic table
        wAddDiagCode            : WORD;         追加診断コード。詳細は AXL_CAN_COMM Diagnostic table
        xDone                   : BOOL;         TRUE : コマンド成功（1 周期のみ）
(* 入出力引数 *)
        arrFilter11BitRanges    : CAN_ARR_11BitFilterRange;
                                                11bit ID のフィルタパラメータ配列
END_STRUCT;
```

2.5 EXA_UDT_AXL_CAN_PARA29 : STRUCT　　Function Block AXL_CAN_Para29 入出力引数
AXL_CAN_Para29: AXL F IF CAN 1H の 29bit
フィルタの設定を行う

(* 入力引数 *)

xActivate	: BOOL;	立ち上がりエッジで起動。FALSE で不起動
xReset	: BOOL;	立ち上がりエッジでリセット
xResetParam	: BOOL;	TRUE : 全パラメータを初期化
bFilter29BitMode	: BYTE;	29bit フィルタモード 0: 全て受信, 1: 全てブロック, 2: フィルタ有効(受信)、3: フィルタ有効(送信)

(* 出力引数 *)

xActive	: BOOL;	FALSE: インアクティブ TRUE: アクティブ（起動後アクティブになるのを要確認）
xError	: BOOL;	TRUE : エラー発生。詳細は wDiagCode、wAddDIagCode 参照
wDiagCode	: WORD;	診断コード。詳細は AXL_CAN_COMM Diagnostic table
wAddDiagCode	: WORD;	追加診断コード。詳細は AXL_CAN_COMM Diagnostic table
xDone	: BOOL;	TRUE : コマンド成功（1 周期のみ）

(* 入出力引数 *)

arrFilter29BitRanges	: CAN_ARR_29BitFilterRange;	29bit ID のフィルタパラメータ配列

END_STRUCT;

2.6 EXA_UDT_ASYNCOM_AXL : STRUCT　　Function Block AsynCom_AXL 入出力引数

AsynCom_AXL: Axioline とのパラメータ交換で必要な PDI コマンドを実施する AXL_CAN_Para,AXL_CAN_Para11(29)が使用

```
(* 入力引数 *)
        wSlot                   : WORD;         モジュール番号
        bSubSlot                : BYTE;         モジュール・サブ番号
(* 出力引数 *)
        dwDiagCodeConnect       : DWORD;        不使用
        dwDiagCodeRead          : DWORD;        PDI_READ のエラーコード
        dwDiagCodeWrite         : DWORD;        PDI_WRITE のエラーコード
END_STRUCT;
```

2.7 AXL_CAN_UDT_DIAGSTATE : STRUCT　　Function Block AXL_CAN_Param の出力引数 srDiagState の構造

uiErrorNumber	: UINT;	エラー番号
usiPrio	: USINT;	優先順位 0x0:エラーなし、0x1:エラー、0x2:警告 0x81:中止エラー、0x82 中止警告
usiChannel	: USINT;	チャネル/グループ/モジュール 0x0:エラーなし、0xFF:全モジュール
uiErrorCode	: UINT;	エラーコード
usiAddInfo	: USINT;	追加情報
strErrorMessage	: STRING;	エラーメッセージ

```
    END_STRUCT
```

2.8 ASYN_UDT_COM : STRUCT　　Function Block AsynCom_AXL が使用するデータ構造

ユーザープログラムでの直接アクセス不要。AXL_COM_Para, AXL_COM_Para11(29)が使用

```
                xActivate               : BOOL;
                xOccupied               : BOOL;
                xActive                 : BOOL;
                xError                  : BOOL;
                xAutoConfirm            : BOOL;
                xEnableErrorMsg         : BOOL;
                xNewErrorMsg            : BOOL;
                bBusSystem              : BYTE;
                iMaxRetriesPerMinute    : INT;
                tTimeout                : TIME;
                dwNodeID                : DWORD;
                dwDiagCodeConnect       : DWORD;
                udtConnect              : ASYN_UDT_CONNECT;
                udtRead                 : ASYN_UDT_READ;
                udtWrite                : ASYN_UDT_WRITE;
                udtTest                 : ASYN_UDT_TEST;
                udtDiag                 : ASYN_UDT_DIAG;
                iState                  : INT;
```

END_STRUCT

2.9 CAN_UDT_DATA : STRUCT　　CAN の送受信データを入れるデータ構造
udtExample のメンバ udtCanData として
ExampleMachine で直接設定読み出しする

xTx :	BOOL;	送信モード、受信モードのトグルスイッチ
xActivate :	BOOL;	AXL_CAN_COMM 起動
xActive :	BOOL;	AXL_CAN_COMM アクティブ
xSend :	BOOL;	AXL_CAN_COMM で xSend(送信)がセットされている
arrMessagesSend :	CAN_ARR_MESSAGES_0_199;	送信メッセージ配列
arrMessagesReceive :	CAN_ARR_MESSAGES_0_199;	受信メッセージ配列
arrMessagesSendHigh :	CAN_ARR_MESSAGES_0_3;	高優先度送信メッセージ配列
udtLastMessages :	CAN_UDT_LAST_MESSAGES;	最新の送信/受信メッセージ

END_STRUCT

2.10 CAN_ARR_MESSAGES_0_199 : ARRAY[0..199] OF CAN_UDT_MESSAGE;　　CAN メッセージ配列

2.11 CAN_ARR_MESSAGES_0_3 : ARRAY[0..3] OF CAN_UDT_MESSAGE;　　高優先度 CAN メッセージ配列

2.12 CAN_UDT_LAST_MESSAGES : STRUCT　　最新の送受信 CAN メッセージ

udtLastSendMessage : CAN_UDT_MESSAGE;
udtLastSendHighMessage : CAN_UDT_MESSAGE;
udtLastReceiveMessage : CAN_UDT_MESSAGE;
END_STRUCT

2.13 CAN_UDT_MESSAGE : STRUCT　　1 つの CAN メッセージを包含する構造体

xUsed :	BOOL;	TRUE: メッセージの処理が必要場合
diID :	DINT;	CAN メッセージの ID
iDLC :	INT;	データ長
udiSequence :	UDINT;	受信した同一 ID のメッセージ数（Receive-mode 1 の時）
xFrameFormat :	BOOL;	TRUE: 29 ビット ID、FALSE : 11 ビット ID
xFrameType :	BOOL;	TRUE: RTR （リモートフレーム）
arrData :	CAN_ARR_B_1_8;	CAN メッセージペイロード部分

END_STRUCT

2.14 CAN_ARR_B_1_8 : ARRAY[1..8] OF BYTE;　　CAN メッセージペイロード部分

2.15 CAN_ARR_11BitFilterRange : ARRAY[0..59] OF CAN_UDT_11BitFilter;　　11bit フィルタ配列

2.16 CAN_ARR_29BitFilterRange : ARRAY[0..29] OF CAN_UDT_29BitFilter;　　29bit フィルタ配列

2.17 CAN_UDT_11BitFilter : STRUCT　　11bit フィルター範囲

uiFrom	: UINT;
uiTo	: UINT;

END_STRUCT

2.18 CAN_UDT_29BitFilter : STRUCT　　29bit フィルター範囲

udiFrom	: UDINT;
udiTo	: UDINT;

END_STRUCT

3. CAN Function Block 診断コード

3.1 AXL_CAN_COMM Diagnostic table

AXL_CAN_COMM の出力引数 wDiagCode, wAddDiagCode で想定される値の説明。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000		Function block is in regular operation
.	16#0000	No execution.
.	16#0001	Send mode executed.
.	16#0002	Receive mode executed.
16#C414	16#0001	Sending timeout.
16#C420	16#0001	Receive array full.
16#C900	16#0001	Index out of range. Please check the input process data.

udtExample.udtAXL_CAN_COMM_X1.wDiagCode、udtExample.udtAXL_CAN_COMM_X2.wDiagCode
udtExample.udtAXL_CAN_COMM_X1.wAddDiagCode、udtExample.udtAXL_CAN_COMM_X2.wAddDiagCode
でアクセスされている。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Funciton Block がインアクティブ
16#8000		Function block は起動中
.	16#0000	実行していない

.	16#0001	送信モードで実行中.
.	16#0002	受信モードで実行中.
16#C414	16#0001	送信タイムアウト
16#C420	16#0001	Receive-array フル
16#C900	16#0001	Index 範囲外 input process data.を再確認

3.2 AXL_CAN_Para Diagnostic table

AXL_CAN_Param の出力引数 wDiagCode, wAddDiagCode で想定される値の説明。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000		Function block is in regular operation
.	16#0000	No execution.
.	16#0001	Send mode executed.
.	16#0002	Receive mode executed.
16#C414	16#0001	Sending timeout.
16#C420	16#0001	Receive array full.
16#C900	16#0001	Index out of range. Please check the input process data.

udtExample.udtAXL_CAN_Para.wDiagCode、udtExample.udtAXL_CAN_Para.wAddDiagCode
でアクセスされている。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block がインアクティブ
16#8000		Function block は起動中
16#C010	16#0000	通信 function block でエラー発生!
16#C011	16#0000	タイムアウト 通信 function block 起動不可!
16#C020	16#0000	品番読み出し中エラー! 通信 function block 利用不可!

16#C025	16#0000	エラー! 誤ったモジュールタイプとの接続
16#C026	16#0000	品番読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C027	16#0000	品番読み出し中のエラー。ウォッチドッグ時間超過。. 通信 function block 応答なし!
16#C030	16#0000	ファームウエアバージョン読み出し中のエラー。!
16#C035	16#0000	ファームウエアバージョン読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C036	16#0000	ファームウエアバージョン読み出し中のエラー。ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C040	16#0000	ロケーション読み出し中のエラー! 通信 function block 利用不可!
16#C045	16#0000	ロケーション読み出し中通信 function block エラー! dwAddDiagCode = dwAddDiagCode =非同期通信用 function block からの診断コード
16#C046	16#0000	ロケーション読み出し中のエラー。ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C050	16#0000	装置名読み出し中のエラー! 通信 function block 利用不可!

16#C055	16#0000	装置名読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C056	16#0000	装置名読み出し中のエラー。ウォッチドッグ時間超過。. 通信 function block 応答なし!
16#C060	16#0000	ビットレート読み出し中のエラー! 通信 function block 利用不可!
16#C065	16#0000	ビットレート読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C066	16#0000	ビットレート読み出し中のエラー。ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C101	16#0000	エラー! 不正なビットレート!
16#C200	16#0000	ビットレート読み出し中のエラー! 通信 function block 利用不可!
16#C210	16#0000	ビットレート読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C211	16#0000	ビットレート読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C300	16#0000	ビットレート読み出し中のエラー! 通信 function block 利用不可!

16#C310	16#0000	診断状態読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C311	16#0000	診断状態読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C400	16#0000	ロケーション設定中のエラー! 通信 function block 利用不可!
16#C410	16#0000	ロケーション設定中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C411	16#0000	ロケーション設定中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C430	16#0000	ロケーション読み出し中のエラー! 通信 function block 利用不可!
16#C440	16#0000	ファームウエアバージョン読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C441	16#0000	ロケーション読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C450	16#0000	エラー! ロケーション設定不可! 設定しようとした値とモジュールから読み出した値が異なる。
16#C500	16#0000	装置名設定中のエラー! 通信 function block 利用不可!

16#C510	16#0000	装置名設定中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C511	16#0000	装置名設定中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C530	16#0000	装置名読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C540	16#0000	装置名読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C541	16#0000	装置名読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C550	16#0000	エラー! 装置名設定不可! 設定しようとした値とモジュールから読み出した値が異なる。
16#C600	16#0000	診断状態リセット中のエラー! 通信 function block 利用不可!
16#C610	16#0000	診断状態リセット中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C611	16#0000	診断状態リセット中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C700	16#0000	全パラメータのデフォルト値設定中のエラー! 通信 function block 利用不可!

16#C710	16#0000	全パラメータのデフォルト値設定中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C711	16#0000	全パラメータのデフォルト値設定中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C730	16#0000	ロケーション読み出し中エラー!
16#C740	16#0000	ロケーション読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C741	16#0000	ロケーション読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C750	16#0000	装置名読み出し中のエラー! 通信 function block 利用不可!
16#C760	16#0000	装置名読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C761	16#0000	装置名読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C770	16#0000	ビットレート読み出し中のエラー! 通信 function block 利用不可!
16#C780	16#0000	ビットレート読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード

16#C781	16#0000	ビットレート読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C800	16#0000	ビットレート設定中のエラー! 通信 function block 利用不可!
16#C810	16#0000	ビットレート設定中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C811	16#0000	ビットレート設定中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C830	16#0000	ビットレート読み出し中のエラー! 通信 function block 利用不可!
16#C840	16#0000	ビットレート読み出し中の通信 function block エラー! dwAddDiagCode =非同期通信用 function block からの診断コード
16#C841	16#0000	ビットレート読み出し中のエラー!ウォッチドッグ時間超過。.通信 function block 応答なし!
16#C850	16#0000	エラー!ビットレート設定不可! 設定しようとした値とモジュールから読み出した値が異なる。
16#C900	16#0000	応答を文字列に変換中のエラー(BUF_TO_STRING エラー)

3.3 AXL_CAN_Para11 Diagnostic table

AXL_CAN_Para11 の出力引数 wDiagCode で想定される値の説明。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000		Function block is in regular operation
.	16#0000	No execution.
.	16#0001	Send mode executed.
.	16#0002	Receive mode executed.
16#C414	16#0001	Sending timeout.
16#C420	16#0001	Receive array full.
16#C900	16#0001	Index out of range. Please check the input process data.

udtExample.udtAXL_CAN_Para11.wDiagCode
でアクセスされている。（wAddDiagCode は未使用）

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000		Function block is in regular operation
16#C010	16#0000	Error in communication function block!
16#C011	16#0000	Timeout! Communication function block could not be activated!
16#C020	16#0000	Error during reading or number occurred! Communication function block is not available!

16#C025	16#0000	Error! Wrong module type connected!
16#C026	16#0000	Error in communication function block during reading order number occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C027	16#0000	Error during reading order number occurred. Watchdog time exceeded. Communication function block does not response!
16#C030	16#0000	Error during reading firmware version occurred!
16#C035	16#0000	Error in communication function block during reading firmware version occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C036	16#0000	Error during reading firmware version occurred. Watchdog time exceeded. Communication function block does not response!
16#C040	16#0000	Error during reading location occurred! Communication function block is not available!
16#C045	16#0000	Error in communication function block during reading location occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C046	16#0000	Error during reading location occurred. Watchdog time exceeded. Communication function block does not response!
16#C050	16#0000	Error during reading equipment identifier occurred! Communication function block is not available!

16#C055	16#0000	Error in communication function block during reading equipment identifier occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C056	16#0000	Error during reading equipment identifier occurred. Watchdog time exceeded. Communication function block does not response!
16#C060	16#0000	Error during reading bit rate occurred! Communication function block is not available!
16#C065	16#0000	Error in communication function block during reading bit rate occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication!
16#C066	16#0000	Error during reading bit rate occurred. Watchdog time exceeded. Communication function block does not response!
16#C101	16#0000	Error! Invalid bit rate!
16#C200	16#0000	Error during reading bit rate occurred! Communication function block is not available!
16#C210	16#0000	Error in communication function block during reading bit rate occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C211	16#0000	Error during reading bit rate occurred. Watchdog time exceeded. Communication function block does not response!
16#C300	16#0000	Error during reading diagnosis state occurred! Communication function block is not available!

16#C310	16#0000	Error in communication function block during reading diagnosis state occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C311	16#0000	Error during reading diagnosis state occurred. Watchdog time exceeded. Communication function block does not response!
16#C400	16#0000	Error during setting location occurred! Communication function block is not available!
16#C410	16#0000	Error in communication function block during setting location occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C411	16#0000	Error during setting location occurred. Watchdog time exceeded. Communication function block does not response!
16#C430	16#0000	Error during reading location occurred! Communication function block is not available!
16#C440	16#0000	Error in communication function block during reading firmware version occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C441	16#0000	Error during reading location occurred. Watchdog time exceeded. Communication function block does not response!
16#C450	16#0000	Error! Location could not be set! Value to be set and value read back from the module are different!
16#C500	16#0000	Error during setting equipment identifier occurred! Communication function block is not available!

16#C510	16#0000	Error in communication function block during setting equipment identifier occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C511	16#0000	Error during setting equipment identifier occurred. Watchdog time exceeded. Communication function block does not response!
16#C530	16#0000	Error during reading equipment identifier occurred! Communication function block is not available!
16#C540	16#0000	Error in communication function block during reading equipment identifier occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C541	16#0000	Error during reading equipment identifier occurred. Watchdog time exceeded. Communication function block does not response!
16#C550	16#0000	Error! Equipment identifier could not be set! Value to be set and value read back from the module are different!
16#C600	16#0000	Error during resetting diagnosis occurred! Communication function block is not available!
16#C610	16#0000	Error in communication function block during resetting diagnosis occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C611	16#0000	Error during resetting diagnosis occurred. Watchdog time exceeded. Communication function block does not response!
16#C700	16#0000	Error during setting all parameters to default values occurred! Communication function block is not available!

16#C710	16#0000	Error in communication function block during resetting all parameters to default values occurred! dwAddDiagCode = Diagnosis code provided, by function block for asynchronous communication.
16#C711	16#0000	Error during resetting all parameters to default values occurred. Watchdog time exceeded. Communication function block does not response!
16#C730	16#0000	Error during reading location occurred!
16#C740	16#0000	Error in communication function block during reading location occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C741	16#0000	Error during reading location occurred. Watchdog time exceeded. Communication function block does not response!
16#C750	16#0000	Error during reading equipment identifier occurred! Communication function block is not available!
16#C760	16#0000	Error in communication function block during reading equipment identifier occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C761	16#0000	Error during reading equipment identifier occurred. Watchdog time exceeded. Communication function block does not response!
16#C770	16#0000	Error during reading bit rate occurred! Communication function block is not available!

16#C780	16#0000	Error in communication function block during reading bit rate occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C781	16#0000	Error during reading bit rate occurred. Watchdog time exceeded. Communication function block does not response!
16#C800	16#0000	Error during setting bit rate occurred! Communication function block is not available!
16#C810	16#0000	Error in communication function block during setting bit rate occurred! dwAddDiagCode = Diagnosis code provided by function block for asynchronous communication.
16#C811	16#0000	Error during setting bit rate occurred. Watchdog time exceeded. Communication function block does not response!
16#C830	16#0000	Error during reading bit rate occurred! Communication function block is not available!
16#C840	16#0000	Error in communication function block during reading bit rate occurred! dwAddDiagCode = Diagnosis code, provided by function block for asynchronous communication.
16#C841	16#0000	Error during reading bit rate occurred. Watchdog time exceeded. Communication function block does not response!
16#C850	16#0000	Error! Bit rate could not be set! Value to be set and value read back from the module are different!
16#C900	16#0000	Error occured while converting the response to string (BUF_TO_STRING Error).

DiagCode	Description
16#0000	Function block はインアクティブ
16#8000	Function block は起動中
16#C010	通信 function block エラー!
16#C011	タイムアウト! 通信 function block 起動不可!
16#C020	エラー! 不正なフィルターモード値!
16#C030	エラー! フィルター開始パラメータ(From)範囲外! (Max = 2047)
16#C031	エラー! フィルター終了パラメータ(To)範囲外! (Max = 2047)
16#C032	エラー!不正なフィルタパラメータ範囲!
16#C040	品番読み出し中のエラー!
16#C050	エラー! 誤ったモジュール接続(誤ったノード ID)!
16#C051	読み出し中の通信 function block エラー
16#C052	品番読み出し中エラー
16#C100	フィルタモード設定中エラー!
16#C110	フィルタおﾒ一度設定中の通信 function block エラー
16#C111	フィルタモード設定中エラー。ウォッチドッグ時間超過
16#C120	フィルタパラメータ設定エラー!
16#C130	フィルタ設定中の通信 function block エラー
16#C131	ロケーション読み出し中エラー
16#C400	全パラメータのデフォルト値設定中エラー!
16#C410	全パラメータリセット中の通信 function block エラー

16#C411	全パラメータのデフォルト値リセット中エラー

3.4 AXL_CAN_Para29 Diagnostic table

AXL_CAN_Para29 の出力引数 wDiagCode で想定される値の説明。

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000		Function block is in regular operation
.	16#0000	No execution.
.	16#0001	Send mode executed.
.	16#0002	Receive mode executed.
16#C414	16#0001	Sending timeout.
16#C420	16#0001	Receive array full.
16#C900	16#0001	Index out of range. Please check the input process data.

udtExample.udtAXL_CAN_Para29.wDiagCode
でアクセスされている。(wAddDiagCode は未使用)

DiagCode	Description
16#0000	Function block インアクティブ
16#8000	Function block 起動中
16#C010	通信 function block エラー!
16#C011	タイムアウト! 通信 function block 起動不可!
16#C020	エラー!不正なフィルタモード値!
16#C030	エラー! フィルタ開始パラメータ(From)範囲外! (Max = 536870911)

16#C031	エラー! フィルタ終了パラメータ(To)範囲外! (Max = 536870911)
16#C032	エラー!不正なフィルタパラメータ範囲!
16#C040	品番読み出し中のエラー!
16#C050	エラー! 誤ったモジュールの接続(誤ったノード ID)!
16#C051	通信 function block 読み出し中エラー
16#C052	品番読み出し中エラー
16#C100	フィルタモード設定中エラー!
16#C110	通信 function block フィルタモード設定中エラー
16#C111	フィルタモード設定中エラー。ウォッチドッグ時間超過
16#C120	フィルタパラメータ設定中エラー!
16#C130	通信 function block フィルタ設定中エラー
16#C131	ロケーション読み出し中エラー
16#C400	全パラメータのデフォルト値設定中エラー!
16#C410	通信 function block 全パラメータのリセット中エラー
16#C411	全パラメータのデフォルト値リセット中エラー