Application Note

Absolute Encoder Reset

Applicable Products:

Yaskawa SERVOPACKs with CANopen over EtherCAT

Product: Yaskawa SERVOPACKs with CANopen over EtherCAT	**Doc#:** AN.MTN.05.ETHERCAT
Title: Absolute Encoder Reset	

Table of Contents

Product: Yaskawa SERVOPACKs with CANopen over EtherCAT	**Doc#:** AN.MTN.05.ETHERCAT
Title: Absolute Encoder Reset	

1. About This Document

This document contains instructions to command through EtherCAT the Absolute Encoder Multiturn Reset and clear the associated alarm (A.810) from Yaskawa EtherCAT SERVOPACKs, without requiring hardware power cycle.

2. Instructions

1. Issue the Adjusting command for “Absolute encoder reset”.
    - See, “3. Absolute Encoder Multiturn Reset”.
2. Issue the Adjusting command for “Software reset”.
    - See, “4. Software Reset Over EtherCAT”.

3. Absolute Encoder Multiturn Reset

3.1. PLC Example Code

3 separate PLC code sections of example code are provided in "Appendix A: Example PLC Code".

- Pseudocode is provided in "6. Pseudocode".

The example PLC code can also be viewed by using TwinCAT to open the YaskawaFunctions.lib included in Yaskawa.com document number TM.SGDV.01.

The example PLC code was written in Beckhoff TwinCAT 2 which follows IEC61131-C.

- PLCs that follow IEC61131-C may be able to interpret this code, with the exception of the Beckhoff function blocks.
- EtherCAT masters may have their own Function block names; this code may need some modifications to work with other EtherCAT masters.

The method used by this example code is to create an array of USINTs for the data to be sent with an SDO write.

3.2. Designing PLC code

The core functionality is described in the SGDV EtherCAT user's manual.

- For reference, see "5. User's Manual References".
- A breakdown of the Command data format is provided after the User's Manual References.

Wireshark can be used to verify that the data being sent is the correct length.

4. Software Reset Over EtherCAT

4.1. Basic Function Details

For SGDV, refer to the chapter "Object Dictionary" section "Manufacturer Specific Objects" subsection "SERVOPACK Adjusting Command (2710h)" in Yaskawa.com document number SIEPC72082904, or "5. User's Manual References".

In addition to the Executable Adjustments listed, "Software Reset" is also available as follows:

Adjustment	Request Code	Preparation before execution	Processing Time	Execution Conditions
Software Reset	2006H	None	5 s max.*	Adjustment is disabled: • While the servo is ON • While Servo-ON has been requested

* Communications to the SERVOPACK will be lost when execution of the "Software Reset" command completes successfully.

In the instructions "How to Send a Command for Adjustment", after step 3 is performed to execute the adjustment "Software Reset", when the slave station receives the command normally, the communications to the SERVOPACK will be lost during the software reset.
Communications will be re-established when the software reset is complete.

4.2. PLC Sample Code

Pseudocode for the Software Reset Over EtherCAT function is provided in "6. Pseudocode".

5. User's Manual References

Images from Yaskawa.com document number SIEPC72082904.

(7) SERVOPACK Adjusting Command (2710h)

This object should be used for SERVOPACK adjusting services (e.g., Encoder setup, Multi-turn reset and so on). Writes the data into the Sub-Index 1 to start the command execution. Also, reads the Sub-Index 3 to accept the response. If the response is not available when reading Sub-Index 3, the first byte of the reply data could give information about the progress.

Index	Sub	Name	Data Type	Access	PDO Mapping	Value	EEPROM
2710h	0	Number of entries	USINT	RO	No	3	No
	1	Command	STRING [16]	RW	No	Byte 0 to n: Service Request Data A write access to the command data will execute the command	No
	2	Status	USINT	RO	No	0: last command completed, no errors, no reply 1: last command completed, no errors, reply there 2: last command completed, error, no reply 3: last command completed, error, reply there 255: command is executing	No
	3	Reply	STRING [16]	RO	No	Byte 0: as Subindex 2 Byte 1: unused 2 to n: Service Response Data	No

■ Command/Reply data format

Command data (Service Request data)	
Byte	Description
0	Reserved
1	Reserved
2	CCMD (Command code) 00: Read request 01: Write request
3	CSIZE (CDATA length in byte)
4 to 7	CADDRESS (Address)
8 to 15	CDATA (Writing data)

Reply data (Service Response data)	
Byte	Description
0	Status (As Subindex 2)
1	Reserved
2	RCMD (Echo back of CCMD)
3	RSIZE (R_DATA length in byte)
4 to 7	RADDRESS (Echo back of CADDRESS)
8 to 15	RDATA (Read data) /ERROCODE

■ Executable Adjustments

Adjustment	Request Code	Preparation before execution	Processing Time	Execution Conditions
Absolute encoder reset	1008H	Required	5 s max.	When using an incremental encoder, impossible to reset the encoder while the servo is ON.
Automatic offset adjustment of motor current detection signals	100EH	None	5 s max.	Adjustment is disabled: • While the main circuit power supply is OFF • While the servo is ON • While the servomotor is running
Multiturn limit setting	1013H	Required	5 s max.	When using an incremental encoder, the setting is disabled unless A.CC0 (Multiturn limit disagreement) occurs.

■ How to Send an Command for Adjustment

1. Send the following data and set the request code of the adjustment to be executed.
   CCMD = 0001H
   CADDRESS = 2000H
   CSIZE = 0002H
   CDATA = Request code of the adjustment to be executed
   When the slave station receives the command normally, status field will be returned to 1.
   If an error occurs, carry out the operation in step 4 to abort execution.
2. For adjustment that requires a preparation process, send the following data.
   If the preparation before execution is not required, carry out the operation in step 3.
   CCMD = 0001H
   CADDRESS = 2001H
   CSIZE = 0002H
   CDATA = 0002H
   When the slave station receives the command normally, status field will be returned to 1.
   If an error occurs, carry out the operation in step 4 to abort execution.
3. Send the following data to execute adjustment.
   CCMD = 0001H
   CADDRESS = 2001H
   CSIZE = 0002H
   CDATA = 0001H
   When the slave station receives the command normally, status field will be returned to 1.
   If an error occurs, carry out the operation in step 4 to abort execution.
4. Send the following data to abort the execution.
   CCMD = 0001H
   CADDRESS = 2000H
   CSIZE = 0002H
   CDATA = 0000H
   When the slave station receives the command normally, status field will be returned to 1.

Note: If no command can be received in 10 seconds after step1, adjustment operation will be automatically aborted.

Breakdown of:

■ How to Send an Command for Adjustment

1. Send the following data and set the request code of the adjustment to be executed.

CCMD = 0001H
CADDRESS = 2000H
CSIZE = 0002H
CDATA = Request code of the adjustment to be executed
When the slave station receives the command normally, status field will be returned to 1.
If an error occurs, carry out the operation in step 4 to abort execution.

"Send the following data and set the request code of the adjustment to be executed", where the adjustment is "Absolute encoder reset", so the request code is 1008H.

The data to be sent is in the example below as variable "SEND_REQUEST_1008" (from example PLC code).

```
Byte number:                                  0 1 2 3  4     5 6 7     8     9 10 11 12 13 14 15
SEND_REQUEST_1008 : ARRAY [1..16] OF USINT :=  0,0,1,2,00,16#20,0,0,16#08,16#10,0, 0, 0, 0, 0, 0;
                                               | | | |  |     | | |     |     | |  |  |  |  |  |
            * Bytes 0 and 1 are reserved, set to 0 | |  |     | | |     |     | |  |  |  |  |  |
* Byte 2 is the CCMD (Command Code), set to 1 (Write) |  |     | | |     |     | |  |  |  |  |  |
          * Byte 3 is the CSIZE, set to 2 as specified  |     | | |     |     | |  |  |  |  |  |
      * Bytes 4 to 7 are the CADDRESS. Set to 2000H with left as LSB      |     | |  |  |  |  |  |
        That is, Byte 4 is 00, Byte 5 is 20H. (6 and 7 are 0)             |     | |  |  |  |  |  |
                * Bytes 8 to 15 are the CDATA, which is 1008H in this case, with left as LSB.
                  That is, Byte 8 is 08H, Byte 9 is 10H. (10-15 are 0)
```

6. Pseudocode

```
Execute_Absolute_Encoder_Reset() function:
1. (* Check execution conditions for Absolute Encoder Reset: Servo OFF *)
   If ( Operation_Enabled OR Quick_Stop_Active OR Fault_Reaction_Active) Then
     Display error: Execution conditions not met (servo must be off).
     End routine.
2. (* Send the data for the Request code of the adjustment Absolute Encoder Reset *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,00,16#20,0,0,16#08,16#10,0,0,0,0,0,0] )
3. (* Check status for normal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If ( Adjusting_Status = 0  OR  Adjusting_Status = 1 ) Then
        (* Status is: Last command completed. No Error *)
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Run Abort_Adjusting() function.
4. (* Send the data for the Preparation process *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,01,16#20,0,0,16#02,16#00,0,0,0,0,0,0] )
5. (* Check status for normal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If ( Adjusting_Status = 0  OR  Adjusting_Status = 1 ) Then
        (* Status is: Last command completed. No Error *)
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Run Abort_Adjusting() function.
6. (* Send the data to execute the adjustment *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,01,16#20,0,0,16#01,16#00,0,0,0,0,0,0] )
7. (* Check status for normal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If ( Adjusting_Status = 0  OR  Adjusting_Status = 1 ) Then
        (* Status is: Last command completed. No Error *)
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Run Abort_Adjusting() function.
8. (End of Execute_Absolute_Encoder_Reset() function)
```

```
Execute_Software_Reset() function:
1. (* Check execution conditions for Software Reset: Servo OFF *)
   If ( Operation_Enabled OR Quick_Stop_Active OR Fault_Reaction_Active) Then
     Display error: Execution conditions not met (servo must be off).
     End routine.
2. (* Send the data for the Request code of the adjustment Software Reset *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,00,16#20,0,0,16#06,16#20,0,0,0,0,0,0] )
3. (* Check status for normal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If ( Adjusting_Status = 0  OR  Adjusting_Status = 1 ) Then
        (* Status is: Last command completed. No Error *)
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Run Abort_Adjusting() function.
4. (* Send the data to execute the adjustment *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,01,16#20,0,0,16#01,16#00,0,0,0,0,0,0] )
5. (* Check connection and status for abnormal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If (Read_SDO fails) Then
        (* Status is: Communication to the drive has been disconnected – Drive has reset *)
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Run Abort_Adjusting() function.
        (*Or retry Execute_Software_Reset() additional times*)
6. (End of Execute_Software_Reset() function)
```

```
Abort_Adjusting() function:
1. (* Send the data to abort the execution *)
   Write_SDO ( Index    = 2710,
               Subindex = 1,
               Data     = [0,0,1,2,00,16#20,0,0,16#00,16#00,0,0,0,0,0,0] )
2. (* Check status for normal reception of the command *)
   Repeat the following:
   a. Read Adjusting Status: Read_SDO ( Index    = 2710,
                                        Subindex = 2 )
   b. If ( Adjusting_Status = 0  OR  Adjusting_Status = 1 ) Then
        (* Status is: Last command completed. No Error *)
        Display error: Abort succeeded, but Adjusting command failed.
        Terminate the repetition.
   c. If ( Adjusting_Status = 2  OR  Adjusting_Status = 3 ) Then
        (* Status is: Last command completed. Error *)
        Display error: Abort failed and Adjusting command failed.
        Terminate the repetition.
3. (End of Abort_Adjusting() function)
```

Appendix A: **Example PLC Code**

The first section is the In/Out variables for the POU.

The second section is the global variables.

The third section is the execution area of the POU.

A.1 In/Out Variables For The POU

```
FUNCTION_BLOCK Y_AbsoluteEncoderReset

VAR_INPUT
  bExecute        :  BOOL        ;
  sNetId          :  T_AmsNetId  ;
  nSlaveAddr      :  UINT        ;
END_VAR

VAR_OUTPUT
  bDone           :  BOOL        ;
  bBusy           :  BOOL        ;
  bError          :  BOOL        ;
  nErrorID        :  UDINT       ;
END_VAR

VAR
(*Extra Features Var Out*)
  sErrorString    :  T_MaxString ;

(*Internal variables*)
  State           :  UINT  :=  0 ;
  bNotExecute     :  BOOL        ;
  StatusWord      :  UINT        ;
  nErrorIDRead    :  UDINT       ;
  sNetIdRead      :  T_AmsNetId  ;
  nSlaveAddrRead  :  UINT        ;
  StatusAdjusting :  USINT       ;

(*Internal Function Blocks*)
  fbRisingEdge    :  R_TRIG      ;
  fbFallingEdge   :  F_TRIG      ;
  fbSdoReadEx     :  FB_EcCoESdoReadEx    := (bExecute := FALSE, bCompleteAccess := FALSE);
  fbSdoWriteEx    :  FB_EcCoESdoWriteEx   := (bExecute := FALSE, bCompleteAccess := FALSE);
END_VAR
```

A.2 Global Variables

```
VAR_GLOBAL CONSTANT
  SEND_REQUEST_1008 : ARRAY [1..16] OF USINT := 0,0,1,2,00,16#20,0,0,16#08,16#10,0,0,0,0,0,0;
  SEND_REQUEST_100E : ARRAY [1..16] OF USINT := 0,0,1,2,00,16#20,0,0,16#0E,16#10,0,0,0,0,0,0;
  SEND_REQUEST_1013 : ARRAY [1..16] OF USINT := 0,0,1,2,00,16#20,0,0,16#13,16#10,0,0,0,0,0,0;
  SEND_PREPARATION  : ARRAY [1..16] OF USINT := 0,0,1,2,01,16#20,0,0,16#02,16#00,0,0,0,0,0,0;
  SEND_EXECUTE      : ARRAY [1..16] OF USINT := 0,0,1,2,01,16#20,0,0,16#01,16#00,0,0,0,0,0,0;
  SEND_ABORT        : ARRAY [1..16] OF USINT := 0,0,1,2,00,16#20,0,0,16#00,16#00,0,0,0,0,0,0;
END_VAR
VAR_GLOBAL
END_VAR
```

A.3 Execution Area Of The POU

```
fbFallingEdge  (CLK:= bExecute);
IF fbFallingEdge.Q THEN
  bNotExecute  :=  TRUE;
END_IF;

(* Done is TRUE for one cycle when falling edge of Execute has occurred *)
IF bDone AND bNotExecute THEN
  bDone    :=  FALSE;
END_IF;

IF bError AND bNotExecute THEN
  bError        :=  FALSE;
  nErrorId      :=  0;
  nErrorIDRead  :=  0;
END_IF;

CASE State OF
0:
fbRisingEdge    (CLK:= bExecute);
IF fbRisingEdge.Q THEN
  bDone          :=  FALSE;
  bBusy          :=  TRUE;
  bError         :=  FALSE;
  nErrorId       :=  0;
  nErrorIDRead   :=  0;
  sErrorString   :=  '';
  bNotExecute    :=  FALSE;
  sNetIdRead     :=  sNetId;
  nSlaveAddrRead :=  nSlaveAddr;
  State          :=  10;
END_IF

10: (*check conditions for Absolute Reset - check for servo off - doesn't matter the encoder type, since incremental cannot be checked
through ECAT*)
fbSdoReadEx(       (*Get StatusWord*)
  sNetId     := sNetIdRead,
  nSlaveAddr := nSlaveAddrRead ,
  nSubIndex  := 0,
  nIndex     := 16#6041,
  pDstBuf    := ADR(StatusWord),
  cbBufLen   := SIZEOF(StatusWord),
  bExecute   := TRUE     ); (*rising edge required*)
State:=11;
```

```
11: (*Read the statusword to verify correct state for sending the adjusting command*)
fbSdoReadEx();
IF NOT fbSdoReadEx.bBusy THEN
  IF NOT fbSdoReadEx.bError THEN
    fbSdoReadEx(bExecute:= FALSE);
    IF
      NOT((UINT_TO_WORD(StatusWord) AND 2#0000_0000_0110_1111)=2#0000_0000_0000_0111) (*NOT Quick stop active*)
      AND
      NOT((UINT_TO_WORD(StatusWord) AND 2#0000_0010_0111_1111)=2#0000_0010_0011_0111) (*NOT Operation enabled*)
      AND
      NOT((UINT_TO_WORD(StatusWord) AND 2#0000_0000_0100_1111)=2#0000_0000_0000_1111) (*NOT Fault reaction active*)
      THEN   (*servo is off - Required specifically for absolute encoder reset*)
      fbSdoWriteEx( (*Initialize WriteEx*)
        sNetId    := sNetIdRead,
        nSlaveAddr  := nSlaveAddrRead,
        bExecute  := FALSE  ); (*Second time initialized - first time was in declarations*)
      State:=12;
    ELSE
      sErrorString:=CONCAT(CONCAT(
        'Servo is on. StatusWord cannot be x00x_0111(bin), xx1x_x011_0111(bin), or x0xx_1111(bin), but read as ',
        UDINT_TO_STRING(StatusWord)),
        '(dec).  ErrorID from FB, not from ADS.'
      );
      nErrorIDRead:=4110;
      State:=99;
    END_IF
  ELSE
    sErrorString:=CONCAT(CONCAT(
      'Error reading StatusWord, Object: ',
      WORD_TO_STRING(fbSdoReadEx.nIndex)),
      '.  Error occurred using Function Block FB_EcCoESdoReadEx.  ErrorID is from FB_EcCoESdoReadEx.'
    );
    nErrorIDRead:=fbSdoReadEx.nErrId;
    State:=99;
  END_IF
END_IF

12: (*Send the data for the Request*)
fbSdoWriteEx(
  nIndex    := 16#2710,
  nSubIndex := 1,
  pSrcBuf   := ADR(SEND_REQUEST_1008), (*Absolute encoder reset*)
  cbBufLen  := SIZEOF(SEND_REQUEST_1008),
  bExecute  := TRUE);
State:=13;
```

```
13: (*Complete sending the request*)
fbSdoWriteEx();
IF NOT fbSdoWriteEx.bBusy THEN
  IF NOT fbSdoWriteEx.bError THEN
    fbSdoWriteEx(bExecute:= FALSE);
    State:=14;
  ELSE
    sErrorString:='Error sending Absolute Encoder Reset request to object 2710.  Error occurred using Function Block FB_EcCoESdoWriteEx.
ErrorID is from FB_EcCoESdoWriteEx.';
    nErrorIDRead:=fbSdoWriteEx.nErrId;
    State:=99;
  END_IF
END_IF

14:
fbSdoReadEx(       (*Get Status (from 2710:02 USINT)*)
  nSubIndex := 2,
  nIndex    := 16#2710,
  pDstBuf   := ADR(StatusAdjusting),
  cbBufLen  := SIZEOF(StatusAdjusting),
  bExecute  := TRUE);
State:=15;
```

```
15: (*Read the Status to determine which state to perform next*)
fbSdoReadEx();
IF NOT fbSdoReadEx.bBusy THEN
  IF NOT fbSdoReadEx.bError THEN
    fbSdoReadEx(bExecute:= FALSE);
    CASE StatusAdjusting OF
      1:
        State:=20;
      2:  (*Error. Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Request 1008 - failed.';
        nErrorIDRead:=4115;
        State:=50;
      3:   (*Error. Read reply, Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Request 1008 - failed.  Reply data exists in 0x2710:03.';
        nErrorIDRead:=4116;
        State:=50;
      255:
        State:=14;
      ELSE
        sErrorString:=CONCAT(CONCAT(
          'Status state is not valid (must be 1,2,3,255).  Read as ',
          UDINT_TO_STRING(StatusAdjusting)),
          '(dec).  ErrorID from FB, not from ADS.'
        );
        nErrorIDRead:=4111;
        State:=99;
      END_CASE
  ELSE
    sErrorString:='Error reading Adjusting Status, Object 2710:02.  Error occurred using Function Block FB_EcCoESdoReadEx.  ErrorID is from
FB_EcCoESdoReadEx.';
    nErrorIDRead:=fbSdoReadEx.nErrId;
    State:=99;
  END_IF
END_IF

20: (*Continue with Step 2: Send the Preparation*)
(*Preparation is required for Absolute Encoder Reset*)
(*Send the data for the Preparation*)
fbSdoWriteEx(
  nIndex    := 16#2710,
  nSubIndex := 1,
  pSrcBuf   := ADR(SEND_PREPARATION),
  cbBufLen  := SIZEOF(SEND_PREPARATION),
  bExecute  := TRUE);
State:=21;
```

```
21: (*Complete sending the request*)
fbSdoWriteEx();
IF NOT fbSdoWriteEx.bBusy THEN
  IF NOT fbSdoWriteEx.bError THEN
    fbSdoWriteEx(bExecute:= FALSE);
    State:=22;
  ELSE
    sErrorString:='Error sending Preparation request to object 2710.  Error occurred using Function Block FB_EcCoESdoWriteEx.  ErrorID is
from FB_EcCoESdoWriteEx.';
    nErrorIDRead:=fbSdoWriteEx.nErrId;
    State:=99;
  END_IF
END_IF

22:
fbSdoReadEx(       (*Get Status (from 2710:02 USINT)*)
  nSubIndex := 2,
  nIndex    := 16#2710,
  pDstBuf   := ADR(StatusAdjusting),
  cbBufLen  := SIZEOF(StatusAdjusting),
  bExecute  := TRUE);
State:=23;
```

```
23: (*Read the Status to determine which state to perform next*)
fbSdoReadEx();
IF NOT fbSdoReadEx.bBusy THEN
  IF NOT fbSdoReadEx.bError THEN
    fbSdoReadEx(bExecute:= FALSE);
    CASE StatusAdjusting OF
      1:
        State:=30;
      2:  (*Error. Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Preparation - failed.';
        nErrorIDRead:=4117;
        State:=50;
      3:  (*Error. Read reply, Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Preparation - failed.  Reply data exists in 0x2710:03.';
        nErrorIDRead:=4118;
        State:=50;
      255:
        State:=22;
      ELSE
        sErrorString:=CONCAT(CONCAT(
          'Status state is not valid (must be 1,2,3,255).  Read as ',
          UDINT_TO_STRING(StatusAdjusting)),
          '(dec).  ErrorID from FB, not from ADS.'
        );
        nErrorIDRead:=4111;
        State:=99;
      END_CASE
  ELSE
    sErrorString:='Error reading Adjusting Status, Object 2710:02.  Error occurred using Function Block FB_EcCoESdoReadEx.  ErrorID is from
FB_EcCoESdoReadEx.';
    nErrorIDRead:=fbSdoReadEx.nErrId;
    State:=99;
  END_IF
END_IF

30: (*Continue with Step 2: Send the Execute*)
(*Send the data for the Execute*)
fbSdoWriteEx(
  nIndex    := 16#2710,
  nSubIndex := 1,
  pSrcBuf   := ADR(SEND_EXECUTE),
  cbBufLen  := SIZEOF(SEND_EXECUTE),
  bExecute  := TRUE);
State:=31;
```

```
31: (*Complete sending the request*)
fbSdoWriteEx();
IF NOT fbSdoWriteEx.bBusy THEN
  IF NOT fbSdoWriteEx.bError THEN
    fbSdoWriteEx(bExecute:= FALSE);
    State:=32;
  ELSE
    sErrorString:='Error sending Execute request to object 2710.  Error occurred using Function Block FB_EcCoESdoWriteEx.  ErrorID is from
FB_EcCoESdoWriteEx.';
    nErrorIDRead:=fbSdoWriteEx.nErrId;
    State:=99;
  END_IF
END_IF

32:
fbSdoReadEx(       (*Get Status (from 2710:02 USINT)*)
  nSubIndex := 2,
  nIndex    := 16#2710,
  pDstBuf   := ADR(StatusAdjusting),
  cbBufLen  := SIZEOF(StatusAdjusting),
  bExecute  := TRUE);
State:=33;
```

```
33: (*Read the Status to determine which state to perform next*)
fbSdoReadEx();
IF NOT fbSdoReadEx.bBusy THEN
  IF NOT fbSdoReadEx.bError THEN
    fbSdoReadEx(bExecute:= FALSE);
    CASE StatusAdjusting OF
      1:
        State:=40;
      2:  (*Error. Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Execute - failed.';
        nErrorIDRead:=4119;
        State:=50;
      3:   (*Error. Read reply, Issue error, then issue Send Abort*)
        sErrorString:='Adjusting command: Send Execute - failed.  Reply data exists in 0x2710:03.';
        nErrorIDRead:=4120;
        State:=50;
      255:
        State:=32;
      ELSE
        sErrorString:=CONCAT(CONCAT(
          'Status state is not valid (must be 1,2,3,255).  Read as ',
          UDINT_TO_STRING(StatusAdjusting)),
          '(dec).  ErrorID from FB, not from ADS.'
        );
        nErrorIDRead:=4111;
        State:=99;
      END_CASE
  ELSE
    sErrorString:='Error reading Adjusting Status, Object 2710:02.  Error occurred using Function Block FB_EcCoESdoReadEx.  ErrorID is from
FB_EcCoESdoReadEx.';
    nErrorIDRead:=fbSdoReadEx.nErrId;
    State:=99;
  END_IF
END_IF

40: (*Finished Execution*)
  bDone  :=  TRUE;
  bBusy  :=  FALSE;
  State:=0;
```

```
50: (*Step 4: Send Abort (only applies if an error occurs)*)
fbSdoWriteEx(
  nIndex    := 16#2710,
  nSubIndex := 1,
  pSrcBuf   := ADR(SEND_ABORT),
  cbBufLen  := SIZEOF(SEND_ABORT),
  bExecute  := TRUE);
State:=51;

51: (*Complete sending the request*)
fbSdoWriteEx();
IF NOT fbSdoWriteEx.bBusy THEN
  IF NOT fbSdoWriteEx.bError THEN
    fbSdoWriteEx(bExecute:= FALSE);
    State:=52;
  ELSE
    sErrorString:='Error sending Abort request to object 2710.  Error occurred using Function Block FB_EcCoESdoWriteEx.  ErrorID is from
FB_EcCoESdoWriteEx.  Also, error occurred performing Adjusting command, but ErrorID of this second error cannot be provided.';
    nErrorIDRead:=fbSdoWriteEx.nErrId;
    State:=99;
  END_IF
END_IF

52:
fbSdoReadEx(       (*Get Status (from 2710:02 USINT)*)
  nSubIndex := 2,
  nIndex    := 16#2710,
  pDstBuf   := ADR(StatusAdjusting),
  cbBufLen  := SIZEOF(StatusAdjusting),
  bExecute  := TRUE);
State:=53;
```

```
53: (*Read the Status to determine which state to perform next*)
fbSdoReadEx();
IF NOT fbSdoReadEx.bBusy THEN
  IF NOT fbSdoReadEx.bError THEN
    fbSdoReadEx(bExecute:= FALSE);
    CASE StatusAdjusting OF
      1:
        sErrorString:=CONCAT(
        sErrorString,  (*Concatenate with error string from previous error state.*)
        '  Abort succeeded, but requested adjusting command failed.  ErrorID is from FB, not from ADS.'
        );
        State:=99;
      2:
        sErrorString:='Abort failed, and requested adjusting command failed.  ErrorID is from FB, not from ADS.';
        nErrorIDRead:=4113;
        State:=99;
      3:
        sErrorString:='Abort failed, and requested adjusting command failed.  Reply data exists in 0x2710:03.  ErrorID is from FB, not from
ADS.';
        nErrorIDRead:=4114;
        State:=99;
      255:
        State:=52;
      ELSE
        sErrorString:=CONCAT(CONCAT(
          'Status state is not valid (must be 1,2,3,255).  Read as ',
          UDINT_TO_STRING(StatusAdjusting)),
          '(dec).  ErrorID from FB, not from ADS.  Also, error occurred performing Adjusting command, but ErrorID of this second error cannot
be provided.'
        );
        nErrorIDRead:=4111;
        State:=99;
      END_CASE
  ELSE
    sErrorString:='Error reading Adjusting Status, Object 2710:02.  Error occurred using Function Block FB_EcCoESdoReadEx.  ErrorID is from
FB_EcCoESdoReadEx.  Also, error occurred performing Adjusting command, but ErrorID of this second error cannot be provided.';
    nErrorIDRead:=fbSdoReadEx.nErrId;
    State:=99;
  END_IF
END_IF
```

```
99: (*Error State*)
  bBusy    :=  FALSE;
  bError   :=  TRUE;
  nErrorID :=  nErrorIDRead;
  State    :=  0;

ELSE
  sErrorString:=CONCAT(CONCAT(
    'Internal function block error.  Function block state not found.  State read as: ',
    UINT_TO_STRING(State)),
    '.  ErrorID from FB, not from ADS.'
  );
  nErrorIDRead:=4112;
  State:=99;
END_CASE
```