Table of contents

1 Foreword ............................................................................................................................................. 5
  1.1 Notes on the documentation ........................................................................................................ 5
  1.2 Safety instructions ......................................................................................................................... 6
  1.3 Notes on information security ........................................................................................................ 7

2 Overview ................................................................................................................................................ 8

3 Drive reference ST_DriveRef ................................................................................................................ 11

4 Function blocks ..................................................................................................................................... 13
  4.1 General SoE .................................................................................................................................. 13
    4.1.1 FB_SoEReset_ByDriveRef ........................................................................................................ 13
    4.1.2 FB_SoEWritePassword_ByDriveRef ......................................................................................... 14
    4.1.3 FB_SoEExecuteCommand_ByDriveRef ...................................................................................... 15
    4.1.4 Function blocks for commands ................................................................................................. 17
    4.1.5 Function blocks for diagnostics ............................................................................................... 20
    4.1.6 Function blocks for determining current values ........................................................................ 26
  4.2 AX5000 SoE ................................................................................................................................... 31
    4.2.1 Conversion functions ................................................................................................................ 31
    4.2.2 FB_SoEAX5000ReadActMainVoltage_ByDriveRef ................................................................ 32
    4.2.3 FB_SoEAX5000SetMotorCtrlWord_ByDriveRef .................................................................... 34
    4.2.4 FB_SoEAX5000FirmwareUpdate_ByDriveRef ........................................................................ 35
  4.3 IndraDrive Cs ................................................................................................................................ 39
    4.3.1 Conversion functions ................................................................................................................ 39
  4.4 F_GetVersionTcDrive ...................................................................................................................... 40
  4.5 SimplePlcMotion ............................................................................................................................. 40
    4.5.1 FB_CoEDriveEnable .................................................................................................................. 40
    4.5.2 FB_CoEDriveMoveVelocity .................................................................................................... 41
    4.5.3 FB_SoEDriveEnable ................................................................................................................... 43
    4.5.4 FB_SoEDriveMoveVelocity ..................................................................................................... 44

5 Data types .............................................................................................................................................. 46
  5.1 General SoE .................................................................................................................................. 46
    5.1.1 ST_SoE_String .......................................................................................................................... 46
    5.1.2 ST_SoE_StringEx ....................................................................................................................... 46
    5.1.3 List types ................................................................................................................................ 46
  5.2 AX5000 SoE ................................................................................................................................... 47
    5.2.1 E_FwUpdateState ..................................................................................................................... 47
    5.2.2 ST_AX5000_C1D for Class 1 diagnosis ..................................................................................... 48
    5.2.3 ST_AX5000DriveStatus ............................................................................................................. 49
    5.2.4 E_AX5000_DriveOpMode ......................................................................................................... 49
  5.3 IndraDrive Cs ................................................................................................................................. 49
    5.3.1 E_IndraDriveCs_DriveOpMode ................................................................................................. 49
    5.3.2 ST_IndraDriveCs_C1D for Class 1 diagnosis ............................................................................ 50
    5.3.3 ST_IndraDriveCsDriveStatus .................................................................................................. 50
  5.4 SERCOS .......................................................................................................................................... 51
    5.4.1 E_SoE_AtribLen ........................................................................................................................ 51
5.4.2  E_SoE_CmdControl .............................................................................................................. 51
5.4.3  E_SoE_CmdState ................................................................................................................ 52
5.4.4  E_SoE_Type ....................................................................................................................... 52

5.5  SimplePlcMotion .................................................................................................................. 53
5.5.1  E_CoEDriveEnableState ................................................................................................. 53
5.5.2  E_DriveMoveVelocityError ............................................................................................ 53
5.5.3  ST_CoEDriveloInterface ................................................................................................. 53
5.5.4  ST_DriveMoveVelocityOptions ...................................................................................... 53
5.5.5  ST_SoEDriveloInterface ................................................................................................. 53

6  Samples .................................................................................................................................... 55
1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement.

No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:


with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2  Safety instructions

Safety regulations

Please note the following safety instructions and explanations! Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚨 DANGER 🚨</td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td>🚨 WARNING 🚨</td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td>🚨 CAUTION 🚨</td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td>📝 NOTE 📝</td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

⚠️ Tip or pointer ⚠️

This symbol indicates information that contributes to better understanding.
1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.
The Tc2_Drive library should no longer be used in newer projects. Please use the Tc2_MC2_Drive library instead (see documentation TwinCAT 3 PLC Lib Tc2_MC2_Drive).

The Tc2_Drive library includes functions and function blocks for SoE drives that access the drive via a drive reference.

### Drive libraries

The three drive libraries Tc2_Drive, Tc2_NcDrive and Tc2_MC2_Drive were developed for different functional purposes, but are almost identical in their functionality. The function blocks of the libraries Tc2_NcDrive and Tc_MC2_Drive form wrapper function blocks around the function blocks of the Tc2_Drive library.

<table>
<thead>
<tr>
<th>Drive library</th>
<th>Use</th>
<th>Access to the drive</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tc2_Drive</strong></td>
<td>Use the Tc2_Drive library if you use the drive entirely from the PLC (i.e. without NC).</td>
<td>The drive is accessed via a drive reference. Within the library, the ST_DriveRef structure is used for this with the NetID as a string. For linking purposes, a structure called ST_PlcDriveRef is also provided with the NetID as a byte array. (See Drive reference ST_DriveRef)</td>
<td>If you want to access parameters in the drive for which no special function block has been implemented, use the function blocks FB_SoERead_ByDriveRef and FB_SoEWrite_ByDriveRef. These function blocks are implemented in the PLC Lib Tc2_EtherCAT in the SoE Interface folder.</td>
</tr>
<tr>
<td><strong>Tc2_NcDrive</strong></td>
<td>Use the Tc2_NcDrive library if you are using the drive via the NC with the Tc2_MC libraries.</td>
<td>The drive is accessed via the NC axis structure (NC_TO_PLC). The function blocks of the Tc2_NcDrive library independently determine the access data to the drive (NetID, address and channel number) via the NC axis ID from the NC axis structure.</td>
<td>If you want to access parameters in the drive for which no special function block has been implemented, use the function blocks FB_SoERead and FB_SoEWrite.</td>
</tr>
<tr>
<td><strong>Tc2_MC2_Drive</strong></td>
<td>Use the Tc2_MC2_Drive library if you are using the drive via the NC with the Tc2_MC library.</td>
<td>The drive is accessed via the MC2 axis reference (AXIS_REF). The function blocks of the Tc2_MC2_Drive library independently determine the access data to the drive (NetID, address and channel number) via the NC axis ID from the MC2 axis reference.</td>
<td>If you want to access parameters in the drive for which no special function block has been implemented, use the function blocks FB_SoERead and FB_SoEWrite.</td>
</tr>
</tbody>
</table>

**Note** the differences when using the drive libraries with AX5000 and Bosch Rexroth IndraDrive CS (see Samples [55])

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_GetVersionTcDrive</td>
<td>Reads version information from the PLC library. The function will be replaced by the global structure stLibVersion_Tc2_Drive.</td>
</tr>
</tbody>
</table>
## Name

- **F_ConvWordToSTAX5000C1D [› 31]** Converts the C1D word (S-0-0011) of the AX5000 to a structure `ST_AX5000_C1D` for Class 1 diagnosis [› 48]
- **F_ConvWordToSTAX5000DriveStatus [› 32]** Converts the Drive status word (S-0-0135) of the AX5000 into the structure `ST_AX5000DriveStatus` [› 49]
- **F_ConvWordToSTIndraDriveCsC1D [› 39]** Converts the C1D word (S-0-0011) of the IndraDrive Cs into the structure `ST_IndraDriveCs_C1D` for Class 1 diagnosis [› 50]
- **F_ConvWordToSTIndraDriveCsDriveStatus [› 39]** Converts the drive status word (S-0-0135) of the IndraDrive Cs into a structure `ST_IndraDriveCsDriveStatus` [› 50]

### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FB_SoEReset_ByDriveRef [› 13]</strong></td>
<td>Resets the drive (S-0-0099).</td>
</tr>
<tr>
<td><strong>FB_SoEWritePassword_ByDriveRef [› 14]</strong></td>
<td>Sets the drive password (S-0-0267).</td>
</tr>
<tr>
<td><strong>FB_SoEExecuteCommand_ByDriveRef [› 15]</strong></td>
<td>Executes a command.</td>
</tr>
<tr>
<td><strong>FB_SoEReadDiagMessage_ByDriveRef [› 20]</strong></td>
<td>Reads the diagnostic message (S-0-0095).</td>
</tr>
<tr>
<td><strong>FB_SoEReadDiagNumber_ByDriveRef [› 21]</strong></td>
<td>Reads the diagnostic number (S-0-0390).</td>
</tr>
<tr>
<td><strong>FB_SoEReadDiagNumberList_ByDriveRef [› 22]</strong></td>
<td>Reads the diagnostic number list (up to 30 entries) (S-0-0375).</td>
</tr>
<tr>
<td><strong>FB_SoEReadClassXDiag_ByDriveRef [› 24]</strong></td>
<td>Reads Class 1 diagnosis (S-0-0011) ... Class 3 diagnosis (S-0-0013).</td>
</tr>
<tr>
<td><strong>FB_SoEWriteCommandControl_ByDriveRef [› 17]</strong></td>
<td>Sets the Command Control.</td>
</tr>
<tr>
<td><strong>FB_SoEReadCommandState_ByDriveRef [› 18]</strong></td>
<td>Checks the command status.</td>
</tr>
<tr>
<td><strong>FB_SoERead_ByDriveRef</strong></td>
<td>Reads a parameter (see PLC Lib Tc2_EtherCAT).</td>
</tr>
<tr>
<td><strong>FB_SoEWrite_ByDriveRef</strong></td>
<td>Writes a parameter (see PLC Lib Tc2_EtherCAT).</td>
</tr>
<tr>
<td><strong>FB_SoEReadAmplifierTemperature_ByDriveRef [› 26]</strong></td>
<td>Reads the drive temperature (S-0-0384).</td>
</tr>
<tr>
<td><strong>FB_SoEReadMotorTemperature_ByDriveRef [› 27]</strong></td>
<td>Reads the motor temperature (S-0-0383).</td>
</tr>
<tr>
<td><strong>FB_SoEReadDcBusCurrent_ByDriveRef [› 30]</strong></td>
<td>Reads the DC bus current (S-0-0381).</td>
</tr>
<tr>
<td><strong>FB_SoEReadDcBusVoltage_ByDriveRef [› 28]</strong></td>
<td>Reads the DC bus voltage (S-0-0380).</td>
</tr>
<tr>
<td><strong>FB_SoEAX5000ReadActMainVoltage_ByDriveRef [› 32]</strong></td>
<td>Reads the mains voltage (P-0-0200).</td>
</tr>
<tr>
<td><strong>FB_SoEAX5000SetMotorCtrlWord_ByDriveRef [› 34]</strong></td>
<td>Sets the Motor Control Word (P-0-0096).</td>
</tr>
<tr>
<td><strong>FB_SoEAX5000FirmwareUpdate_ByDriveRef [› 35]</strong></td>
<td>Executes an automatic firmware update for the AX5000.</td>
</tr>
<tr>
<td><strong>FB_CoEDriveEnable [› 40]</strong></td>
<td>Enables a CoE drive.</td>
</tr>
<tr>
<td><strong>FB_CoEDriveMoveVelocity [› 41]</strong></td>
<td>Generates a simple three-phase velocity profile that can be used to supply a CoE drive directly.</td>
</tr>
<tr>
<td><strong>FB_SoEDriveEnable [› 43]</strong></td>
<td>Enables a SoE drive.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_SoEDriveMoveVelocity [44]</td>
<td>Generates a simple three-phase velocity profile that can be used to supply a SoE drive directly.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT on the development computer</td>
<td>3.1 Build 4016 or higher</td>
</tr>
<tr>
<td>TwinCAT on the Windows CE-Image</td>
<td>3.1 Build 4016 or higher</td>
</tr>
<tr>
<td>TwinCAT on the Windows XP-Image</td>
<td>3.1 Build 4016 or higher</td>
</tr>
</tbody>
</table>
3 Drive reference ST_DriveRef

The drive is accessed via a drive reference. Within the library, the ST_DriveRef structure is used for this with the NetID as a string. Since the NetID usually exists as a byte array at I/O level, a structure ST_PlcDriveRef with the NetID as a byte array is additionally provided. The two structures must be transferred into each other.

Structure ST_PlcDriveRef

```c
TYPE ST_PlcDriveRef :
STRUCT
  aNetId      : T_AmsNetIdArr; (* AmsNetId (array[0..5] of bytes) of the EtherCAT master device.*)
nSlaveAddr  : UINT; (* Address of the slave device.*)
nDriveNo    : BYTE; (* Drive number*)
nDriveType  : BYTE; (* Drive type*)
END_STRUCT
END_TYPE
```

Structure ST_DriveRef

```c
TYPE ST_DriveRef :
STRUCT
  sNetId      : T_AmsNetId; (* AmsNetId (string(23)) of the EtherCAT master device.*)
nSlaveAddr  : UINT; (* Address of the slave device.*)
nDriveNo    : BYTE; (* Drive number*)
nDriveType  : BYTE; (* Drive type*)
END_STRUCT
END_TYPE
```

Mapping the drive reference to the PLC

The drive reference can be mapped to the PLC in the System Manager. To do this, allocate an instance of the structure ST_PlcDriveRef as AT %I*. Subsequently, link aNetID to netId, nSlaveAddr to port and nDriveNo to Chn0 (A) or Chn1 (B). In the case of multiple-channel drives both channels refer to the same netId and port number, since it is an EtherCAT slave.
Transfer of ST_PlcDriveRef and ST_DriveRef

The function blocks in the library Tc2_Drive use an instance of the structure ST_DriveRef. Unlike the structure ST_PlcDriveRef, NetID is expected to be T_AmsNetId (i.e. STRING(23)). To convert the byte array, use the F_CreateAmsNetId() function of the PLC Lib Tc2_System.

```
stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
```
4 Function blocks

4.1 General SoE

4.1.1 FB_SoEReset_ByDriveRef

The drive (S-0-0099) can be reset with the function block FB_SoEReset_ByDriveRef. In the case of multiple-channel devices if necessary, both channels will have to perform a reset. The timeout time must be 10 s, as the reset can take up to 10 s depending on the error. An NC reset will not be performed.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time (10 s) allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>
Sample

```cpp
fbSoEReset : FB_SoEReset_ByDriveRef;
bSoEReset : BOOL;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bSoEReset AND NOT bInit THEN
    fbSoEReset(stDriveRef := stDriveRef, bExecute := TRUE, tTimeout := DEFAULT_ADS_TIMEOUT);
    IF NOT fbSoEReset.bBusy THEN
        fbSoEReset(stDriveRef := stDriveRef, bExecute := FALSE);
        bSoEReset := FALSE;
    END_IF
END_IF
```

### 4.1.2 FB_SoEWritePassword_ByDriveRef

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: Drive reference ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time (10 s) allowed for the execution of the function block.</td>
</tr>
<tr>
<td>sPassword</td>
<td>ST_SoE_String</td>
<td>Password as a Sercos string</td>
</tr>
</tbody>
</table>

Using the FB_SoEWritePassword_ByDriveRef function block the drive password (S-0-0267) can be set.

**Inputs**

```cpp
VAR_INPUT
    stDriveRef : ST_DriveRef;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
sPassword : ST_SoE_String;
END_VAR
```

**Outputs**

```cpp
VAR_OUTPUT
    bBusy : BOOL;
bError : BOOL;
iAdsErrId : UINT;
iSercosErrId : UINT;
END_VAR
```
### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>

#### Sample

```plaintext
fbWritePassword : FB_SoEWritePassword_ByDriveRef;
bWritePassword : BOOL;
sPassword : ST_SoE_String;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
        END_IF
END_IF

IF bWritePassword AND NOT bInit THEN
    fbWritePassword(
        stDriveRef := stDriveRef,
        bExecute := TRUE,
        tTimeout := DEFAULT_ADS_TIMEOUT,
        sPassword := sPassword
    );
    IF NOT fbWritePassword.bBusy THEN
        fbWritePassword(stDriveRef := stDriveRef, bExecute := FALSE);
        bWritePassword := FALSE;
    END_IF
END_IF
```

### 4.1.3 FB_SoEExecuteCommand_ByDriveRef

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>nIdn</td>
<td>WORD</td>
<td></td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td></td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td></td>
</tr>
</tbody>
</table>

With the FB_SoEExecuteCommand_ByDriveRef function block a command can be executed.
## Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nIdn</td>
<td>WORD</td>
<td>Parameter number to which FB_SoEExecuteCommand_ByDriveRef refers, e.g. &quot;P_0_IDN + 160&quot; for P-0-0160</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time (10 s) allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

### Outputs

```plaintext
VAR_OUTPUT
    bBusy           : BOOL;
    bError          : BOOL;
    iAdErrId        : UINT;
    iSercosErrId    : UINT;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>

### Sample

```plaintext
fbExecuteCommand : FB_SoEExecuteCommand_ByDriveRef;
bExecuteCommand  : BOOL;
nIdn : WORD;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bExecuteCommand AND NOT bInit THEN
    nIdn := P_0_IDN + 160;
    fbExecuteCommand(stDriveRef := stDriveRef, bExecute := TRUE, tTimeout := DEFAULT_ADS_TIMEOUT, nIdn := nIdn,);
    IF NOT fbExecuteCommand.bBusy THEN
        fbExecuteCommand(stDriveRef := stDriveRef, bExecute := FALSE);
        bExecuteCommand := FALSE;
    END_IF
END_IF
```
4.1.4 Function blocks for commands

4.1.4.1 FB_SoEWriteCommandControl_ByDriveRef

With the FB_SoEWriteCommandControl_ByDriveRef function block a command can either be prepared, started or aborted.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>nIdn</td>
<td>WORD</td>
<td>Parameter number to which FB_SoEReadCommandState_ByDriveRef refers, e.g. &quot;P_0_IDN + 23&quot; for P-0-0023.</td>
</tr>
<tr>
<td>eCmdControl</td>
<td>E_SoE_CmdControl</td>
<td>Indicates, if a command should be prepared (eSoE_CmdControl_Set := 1), executed (eSoE_CmdControl_SetAndEnable := 3) or aborted (eSoE_CmdControl_Cancel := 0).</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time (10 s) allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>
4.1.4.2  FB_SoEReadCommandState_ByDriveRef

With the FB_SoEReadCommandState_ByDriveRef function block the execution of the command can be checked.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef[11])</td>
</tr>
<tr>
<td>nIdn</td>
<td>WORD</td>
<td>Parameter number to which FB_SoEReadCommandState_ByDriveRef refers, e.g. &quot;P_0_IDN + 160&quot; for P-0-0160</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time (10 s) allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

VAR_OUTPUT

- **bBusy**: BOOL
- **bError**: BOOL
- **eCmdState**: E_SoE_CmdState
- **iAdsErrId**: UINT
- **iSercosErrId**: UINT

**Name** | **Type** | **Description** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>eCmdStat</td>
<td>eSoE_CmdState</td>
<td>Returns the command status.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>

**Sample**

```plaintext
fbReadCommandState : FB_SoEReadCommandState_ByDriveRef;
bReadCommandState : BOOL;
nIdn : WORD;
eCmdState : E_SoE_CmdState;
stPlcDriveRef AT %I*: ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
  stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
  stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
  stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
  stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
  IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
    bInit := FALSE;
  END_IF
END_IF

IF bReadCommandState AND NOT bInit THEN
  nIdn := P_0_IDN + 160;
  fbReadCommandState(
    stDriveRef := stDriveRef,
    bExecute := TRUE,
    tTimeout := DEFAULT_ADS_TIMEOUT,
    nIdn := nIdn,
    eCmdState => eCmdState
  );
  IF NOT fbReadCommandState.bBusy THEN
    fbReadCommandState(stDriveRef := stDriveRef, bExecute := FALSE);
    bReadCommandState := FALSE;
  END_IF
END_IF
```
4.1.5 Function blocks for diagnostics

4.1.5.1 FB_SoEReadDiagMessage_ByDriveRef

With the FB_SoEReadDiagMessage_ByDriveRef function block the diagnosis message can be read as a Sercos String (S-0-0095).

**Inputs**

```plaintext
VAR_INPUT
  stDriveRef : ST_DriveRef;
  bExecute   : BOOL;
  tTimeout   : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

```plaintext
VAR_OUTPUT
  bBusy            : BOOL;
  bError           : BOOL;
  iAdsErrId        : UINT;
  iSercosErrId     : UINT;
  dwAttribute      : DWORD;
  sDiagMessage     : ST_SoE_String;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Returns the attributes of the Sercos parameter.</td>
</tr>
<tr>
<td>sDiagMessage</td>
<td>ST_SoE_String</td>
<td>Returns the diagnosis message.</td>
</tr>
</tbody>
</table>
Sample

```plaintext
function fbDiagMessage : FB_SOEReadDiagMessage_ByDriveRef;
bDiagMessage : BOOL;
sDiagMessage : ST_SOE_String;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

if bInit THEN
  stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
  IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
    bInit := FALSE;
  END_IF
END_IF

IF bDiagMessage AND NOT bInit THEN
  fbDiagMessage(
    stDriveRef := stDriveRef,
    bExecute := TRUE,
    tTimeout := DEFAULT_ADS_TIMEOUT,
    sDiagMessage => sDiagMessage
  );
  IF NOT fbDiagMessage.bBusy THEN
    fbDiagMessage(
      stDriveRef := stDriveRef,
      bExecute := FALSE
    );
    bDiagMessage := FALSE;
  END_IF
END_IF
```

### 4.1.5.2 FB_SOEReadDiagNumber_ByDriveRef

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

With the `FB_SOEReadDiagNumber_ByDriveRef` function block the current diagnostic number can be read as UDINT (S-0-0390).
### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Returns the attributes of the Sercos parameter.</td>
</tr>
<tr>
<td>iDiagNumber</td>
<td>UDIIT</td>
<td>Returns the current diagnostic number.</td>
</tr>
</tbody>
</table>

#### Sample

```plaintext
fbDiagNumber : FB_SoEReadDiagNumber_ByDriveRef;

bDiagNumber : BOOL;
iDiagNumber : UDINT;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bDiagNumber AND NOT bInit THEN
    fbDiagNumber(stDriveRef := stDriveRef, bExecute := TRUE,
                 tTimeout := DEFAULT_ADS_TIMEOUT,
                 iDiagNumber => iDiagNumber);
    IF NOT fbDiagNumber.bBusy THEN
        fbDiagNumber(stDriveRef := stDriveRef, bExecute := FALSE);
        bDiagNumber := FALSE;
    END_IF
END_IF
```

### 4.1.5.3 FB_SoEReadDiagNumberList_ByDriveRef

With the FB_SoEReadDiagNumberList_ByDriveRef function block a history of the diagnosis numbers can be read as list (S-0-0375).
Inputs

VAR_INPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
<tr>
<td>piDiagNumber</td>
<td>POINTER TO ST_SoE_DiagNumList</td>
<td>Pointer to the list of the last max. 30 error numbers. The list consists of the current and maximum number of bytes in the list as well as the 30 list items.</td>
</tr>
<tr>
<td>iSize</td>
<td>UDINT</td>
<td>Size of the list in bytes (as Sizeof())</td>
</tr>
</tbody>
</table>

Outputs

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Returns the attributes of the Sercos parameter.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
fbDiagNumberList := FB_SoEReadDiagNumberList_ByDriveRef;
bDiagNumberList := BOOL;
stDiagNumberList := ST_SoE_DiagNumList;
stPlcDriveRef AT %I* := ST_PlcDriveRef;
stDriveRef := ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bDiagNumberList AND NOT bInit THEN
    fbDiagNumberList{
        stDriveRef := stDriveRef,
        bExecute := TRUE,
```

TE1000 Version: 1.3
With the FB_SoEReadClassXDiag_ByDriveRef function block the current class 1 diagnosis (S-0-0011) ... class 3 diagnosis (S-0-0013) can be read as WORD. For the evaluation of the class 1 diagnosis as structure ST_AX5000_C1D \[48\] there is a conversion function F_ConvWordToSTAX5000C1D \[31\].

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>iDiagClass</td>
<td>USINT</td>
<td>Specifies which diagnosis should be read. The diagnostics parameters may vary from vendor to vendor. All diagnostics parameters (C1D ... C3D) or all bits are not always implemented in them.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td></td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td></td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td></td>
</tr>
<tr>
<td>wClassXDiag</td>
<td>WORD</td>
<td></td>
</tr>
</tbody>
</table>
## Function blocks

### Name | Type | Description
--- | --- | ---
| bBusy | BOOL | This output is set when the function block is activated, and remains set until a feedback is received.
| bError | BOOL | This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
| iAdsErrId | UINT | In the case of a set bError output returns the ADS error code of the last executed command.
| iSercosErrId | UINT | In the case of a set bError output returns the Sercos error of the last executed command.
| dwAttribute | DWORD | Returns the attributes of the Sercos parameter.
| wClassXD | iag | Returns the current Class X diagnosis.

### Sample

```plaintext
fbClassXDiag : FB_SoEReadClassXDiag_ByDriveRef;
bClassXDiag : BOOL;
iDiagClass := USINT := 1;
wClass1Diag := WORD;
stAX5000C1D := ST_AX5000_C1D;
wClass2Diag := WORD;
bInit := BOOL := TRUE;
stPlcDriveRef AT %I* := ST_PlcDriveRef;
stDriveRef := ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := _CreateAmsNetId(stPlcDriveRef.aNetId);
stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bClassXDiag AND NOT bInit THEN
    fbClassXDiag(stDriveRef := stDriveRef, bExecute := TRUE,
iDiagClass := iDiagClass,
tTimeout := DEFAULT_ADS_TIMEOUT);
END_IF

IF NOT fbClassXDiag.bBusy THEN
    fbClassXDiag(stDriveRef := stDriveRef, bExecute := FALSE);
bClassXDiag := FALSE;
CASE fbClassXDiag.iDiagClass OF
1:    wClass1Diag := fbClassXDiag.wClassXDiag;
stAX5000C1D := _ConvWordToSTAX5000C1D(wClass1Diag);
2:    wClass2Diag := fbClassXDiag.wClassXDiag;
END_CASE
END_IF
```

---

**TE1000**  
**Version: 1.3**  
**Page 25**
4.1.6 Function blocks for determining current values

4.1.6.1 FB_SoEReadAmplifierTemperature_ByDriveRef

With the FB_SoEReadAmplifierTemperature_ByDriveRef function block the temperature of the drive (S-0-0384) can be read.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Returns the attributes of the Sercos parameter.</td>
</tr>
<tr>
<td>fAmplifierTemperature</td>
<td>REAL</td>
<td>Returns the drive temperature (e.g. 26.2 corresponds to 26.2 °C).</td>
</tr>
</tbody>
</table>
Sample

```plaintext
Sample

Sample

Sample

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

END_IF

4.1.6.2 FB_SoEReadMotorTemperature_ByDriveRef

With the function block FB_SoEReadMotorTemperature_ByDriveRef the temperature of the motor
(S-0-0383) can be read. If the motor does not contain a temperature sensor, this is 0.0, i.e. 0.0 °C.

Inputs

```plaintext
VAR_INPUT
    stDriveRef : ST_DriveRef;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>
## Outputs

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Returns the attributes of the Sercos parameter.</td>
</tr>
<tr>
<td>fMotorTemperature</td>
<td>REAL</td>
<td>Returns the motor temperature (e.g. 30.5 corresponds to 30.5 °C). If the motor does not contain a temperature sensor, this is 0.0, i.e. 0.0 °C.</td>
</tr>
</tbody>
</table>

### Sample

```plaintext
FB_SoEReadMotorTemperature_ByDriveRef
FB_SoEReadDcBusVoltage_ByDriveRef

stPlcDriveRef AT '%I': ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bReadMotorTemp AND NOT bInit THEN
    fbReadMotorTemp(stDriveRef := stDriveRef, bExecute := TRUE, tTimeout := DEFAULT_ADS_TIMEOUT, fMotorTemperature=>fMotorTemperature)
END_IF

IF NOT fbReadMotorTemp.bBusy THEN
    fbReadMotorTemp(stDriveRef := stDriveRef, bExecute := FALSE);
END_IF

END_IF
```

### 4.1.6.3 FB_SoEReadDcBusVoltage_ByDriveRef

```plaintext
FB_SoEReadDcBusVoltage_ByDriveRef

stDriveRef : ST_DriveRef
bExecute : BOOL
fDcBusVoltage : REAL
```
With the FB_SoEReadDcBusVoltage_ByDriveRef function block the DC-Bus voltage of the drive (S-0-0380) can be read.

### Inputs

**VAR_INPUT**

- `stDriveRef`: `ST_DriveRef`;
- `bExecute`: `BOOL`;
- `tTimeout`: `TIME := DEFAULT_ADS_TIMEOUT`;

### Name | Type | Description
--- | --- | ---
 | **stDriveRef** | `ST_DriveRef` | Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: `ST_DriveRef` [11])
 | **bExecute** | `BOOL` | The function block is enabled via a positive edge at this input.
 | **tTimeout** | `TIME` | Maximum time allowed for the execution of the function block.

### Outputs

**VAR_OUTPUT**

- `bBusy`: `BOOL`;
- `bError`: `BOOL`;
- `iAdsErrId`: `UINT`;
- `iSercosErrId`: `UINT`;
- `dwAttribute`: `DWORD`;
- `fDcBusVoltage`: `REAL`;

### Name | Type | Description
--- | --- | ---
 | **bBusy** | `BOOL` | This output is set when the function block is activated, and remains set until a feedback is received.
 | **bError** | `BOOL` | This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
 | **iAdsErrId** | `UINT` | In the case of a set bError output returns the ADS error code of the last executed command.
 | **iSercosErrId** | `UINT` | In the case of a set bError output returns the Sercos error of the last executed command.
 | **dwAttribute** | `DWORD` | Returns the attributes of the Sercos parameter.
 | **fDcBusVoltage** | `REAL` | Supplies the DC bus voltage of the drive.

### Sample

```plaintext
VAR
  bInit : BOOL;
  fbReadDcBusVoltage : FB_SoEReadDcBusVoltage_ByDriveRef;
  bReadDcBusVoltage : BOOL;
  fDcBusVoltage : REAL;
  stPlcDriveRef AT %I* : ST_PlcDriveRef;
  stDriveRef : ST_DriveRef;
END_VAR

IF bInit THEN
  stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
  stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
  stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
  stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
  IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
    bInit := FALSE;
  END_IF
END_IF

IF bReadDcBusVoltage AND NOT bInit THEN
```
Function blocks

```c
fbReadDcBusVoltage(stDriveRef := stDriveRef, bExecute := TRUE,
tTimeout := DEFAULT_ADS_TIMEOUT,
fDcBusVoltage => fDcBusVoltage );

IF NOT fbReadDcBusVoltage.bBusy THEN
  fbReadDcBusVoltage(stDriveRef := stDriveRef, bExecute := FALSE);
  bReadDcBusVoltage := FALSE;
END_IF
END_IF
```

**4.1.6.4  FB_SoEReadDcBusCurrent_ByDriveRef**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

With the FB_SoEAX5000ReadDcBusCurrent_ByDriveRef function block the DC-Bus current (S-0-0381) can be read.

**Inputs**

VAR_INPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**Outputs**

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>fDcBusCurrent</td>
<td>REAL</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>
Function blocks

### Version: 1.3

#### Name | Type | Description
--- | --- | ---
| dwAttribute | DWORD | Returns the attributes of the Sercos parameter. |
| fDcBusCurrent | REAL | Returns the DC bus current (e.g. 2,040 equals 2,040 A). |

**Sample**

```plaintext
fbReadDcBusCurrent : FB_SoEReadDcBusCurrent_ByDriveRef;
bReadDcBusCurrent : BOOL;
fDcBusCurrent : REAL;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bReadDcBusCurrent AND NOT bInit THEN
    fbReadDcBusCurrent(stDriveRef := stDriveRef, bExecute := TRUE, tTimeout := DEFAULT_ADS_TIMEOUT, fDcBusCurrent=>fDcBusCurrent);
    IF NOT fbReadDcBusCurrent.bBusy THEN
        fbReadDcBusCurrent(stDriveRef := stDriveRef, bExecute := FALSE);
        bReadDcBusCurrent := FALSE;
    END_IF
END_IF
```

## 4.2 AX5000 SoE

### 4.2.1 Conversion functions

#### 4.2.1.1 F_ConvWordToSTAX5000C1D

With this function the Class 1 diagnosis FB_SoEReadClassXDiag_ByDriveRef [24] (S-0-0011) can be changed to an ST_AX5000_C1D [48] structure.

**Inputs**

```plaintext
VAR_INPUT
    wClass1Diag : WORD;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wClass1Diag</td>
<td>WORD</td>
<td>Class 1 diagnosis Word from S-0-0011 (see FB_SoEReadClassXDiag_ByDriveRef [24])</td>
</tr>
</tbody>
</table>
4.2.1.2 F_ConvWordToSTAX5000DriveStatus

With this function the Drive status word (S-0-0135) can be changed to a structure ST AX5000DriveStatus [49].

Inputs

VAR_INPUT
  wDriveStatus : WORD;
END_VAR

Return value

FUNCTION F_ConvWordToSTAX5000DriveStatus : ST_AX5000DriveStatus

Name | Type | Description
--- | --- | ---
wDriveStatus | WORD | Drive status word from S-0-0135 (can be read with FB_SoE_Read_ByDriveRef, can be mapped if necessary).

4.2.2 FB_SoEAX5000ReadActMainVoltage_ByDriveRef

With the FB_SoEAX5000ReadActMainVoltage_ByDriveRef function block the current peak value of the mains voltage of the AX5000 (P-0-0200) can be read.

Inputs

VAR_INPUT
  stDriveRef : ST_DriveRef;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

Return value

FUNCTION FB_SoEAX5000ReadActMainVoltage_ByDriveRef

Name | Type | Description
--- | --- | ---
stDriveRef | ST_DriveRef |  
  bExecute | BOOL |  
  tTimeout | TIME |  
  bBusy | BOOL |  
  bError | BOOL |  
  iAdsErrId | UINT |  
  iSercosErrId | UINT |  
  dwAttribute | DWORD |  
  fActualMainVoltage | REAL |  

Inputs

VAR_INPUT
  stDriveRef : ST_DriveRef;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

### Outputs

**VAR_OUTPUT**

- `bBusy` : BOOL;  
- `bError` : BOOL;  
- `iAdsErrId` : UINT;  
- `iSercosErrId` : UINT;  
- `dwAttribute` : DWORD;  
- `fActualMainVoltage` : REAL;

### Sample

```plaintext
fbReadActMainVoltage := FB_SoEAX5000ReadActMainVoltage_ByDriveRef;
bReadActMainVoltage := BOOL;
fActualMainVoltage := REAL;
stPlcDriveRef AT 'I' := ST_PlcDriveRef;
stDriveRef := ST_DriveRef;

IF bInit THEN
    stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
    stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
    stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
    stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
    IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
        bInit := FALSE;
    END_IF
END_IF

IF bReadActMainVoltage AND NOT bInit THEN
    fbReadActMainVoltage(FbReadActMainVoltage := TRUE,
                         bExecute := TRUE,
                         tTimeout := DEFAULT_ADS_TIMEOUT,
                         fActualMainVoltage => fActualMainVoltage);
    IF NOT fbReadActMainVoltage.bBusy THEN
        fbReadActMainVoltage(stDriveRef := stDriveRef, bExecute := FALSE);
        bReadActMainVoltage := FALSE;
    END_IF
END_IF
```
With the FB_SoEAX5000SetMotorCtrlWord_ByDriveRef function block the ForceLock bit (Bit 0) and the ForceUnlock bit in the Motor Control Word (P-0-0096) can be set, in order to activate or release the brake. Normally the brake is automatically controlled via the Enable of the drive.

With the ForceLock-Bit, the brake can be activated independently from the Enable, with the ForceUnlock bit, the brake can be released independently from the Enable. In the case of simultaneously set ForceLock and ForceUnlock, ForceLock (Brake activated) has the higher priority.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
<tr>
<td>bForceLock</td>
<td>BOOL</td>
<td>Activates the brake independently of the enable.</td>
</tr>
<tr>
<td>bForceUnlock</td>
<td>BOOL</td>
<td>Releases the brake independently of the enable.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>This output is set when the function block is activated, and remains set until a feedback is received.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.</td>
</tr>
<tr>
<td>iAdsErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the ADS error code of the last executed command.</td>
</tr>
<tr>
<td>iSercosErrId</td>
<td>UINT</td>
<td>In the case of a set bError output returns the Sercos error of the last executed command.</td>
</tr>
</tbody>
</table>
Sample

```plaintext
fbSetMotorCtrlWord : FB_SoEAX5000SetMotorCtrlWord_ByDriveRef;
fbSetMotorCtrlWord : BOOL;
bForceLock : BOOL;
bForceUnlock : BOOL;
stPlcDriveRef AT %I* : ST_PlcDriveRef;
stDriveRef : ST_DriveRef;

IF bInit THEN
  stDriveRef.sNetId := F_CreateAmsNetId(stPlcDriveRef.aNetId);
  stDriveRef.nSlaveAddr := stPlcDriveRef.nSlaveAddr;
  stDriveRef.nDriveNo := stPlcDriveRef.nDriveNo;
  stDriveRef.nDriveType := stPlcDriveRef.nDriveType;
  IF (stDriveRef.sNetId <> '') AND (stDriveRef.nSlaveAddr <> 0) THEN
    bInit := FALSE;
  END_IF
END_IF

IF bSetMotorCtrlWord AND NOT bInit THEN
  fbSetMotorCtrlWord(
    stDriveRef := stDriveRef,
    bExecute := TRUE,
    tTimeout := DEFAULT_ADS_TIMEOUT,
    bForceLock := bForceLock,
    bForceUnlock := bForceUnlock
  );
  IF NOT fbSetMotorCtrlWord.bBusy THEN
    fbSetMotorCtrlWord(stDriveRef := stDriveRef, bExecute := FALSE);
    bSetMotorCtrlWord := FALSE;
  END_IF
END_IF
```

4.2.4 **FB_SoEAX5000FirmwareUpdate_ByDriveRef**

```
FB_SoEAX5000FirmwareUpdate_ByDriveRef

- stDriveRef ST_DriveRef
- bExecute BOOL
- tTimeout TIME
- sFirmwareVersion STRING(20)
- sFirmwarePath T_MaxString
- sNetIdPC T_AmsNetID
- iReqEcState UINT
- E_FwUpdateState eFwUpdateState
- iLoadProgress INT
- sSelectedFirmwareFile STRING(255)

With the FB_SoEAX5000FirmwareUpdate_ByDriveRef function block the Firmware of the AX5000 can be checked and automatically changed to a given version (Revision and Build) or to the current Build of the configured revision.

For the update:

- the configured slave type is determined, e.g. AX5103-0000-0010.
- the current slave with the specified slave address is determined, e.g. AX5103-0000-0009.
- the current slave firmware is determined, e.g. v1.05_b0009.
- a comparison of the configuration and the found slave regarding number of channels, current, revision and firmware is made.
- the name of the required firmware file is determined and a search for the file performed.
- the firmware update is executed (if necessary).
- the current slave with the specified slave address is determined again.
- the slave is switched to the predefined EtherCAT state.

A successful update ends with eFwUpdateState = eFwU_FwUpdateDone.

If the update is not required, this is signaled via eFwUpdateState = eFwU_NoFwUpdateRequired.
The firmware is updated via the specified channel (A = 0 or B = 1) from stDriveRef. In the case of two-channel devices only one of the two channels can be used. The other channel signals eFwUpdateState = eFwU_UpdateViaOtherChannelActive or eFwUpdateState = eFwU_UpdateViaOtherChannel.

During the firmware update (eFwUpdateState = eFwU_FwUpdateInProgress), iLoadProgress signals the progress in percent.

**NOTE**

**Faulty update due to interruptions**

Interruptions during the update may result in it not being executed or executed incorrectly. Afterwards, the drive amplifier may no longer be usable without the appropriate firmware.

The rules during the update are:

- The PLC and TwinCAT must not be stopped.
- The EtherCAT connection must not be interrupted.
- The AX5000 must not be switched off.

**Inputs**

```plaintext
VAR_INPUT
   stDriveRef : ST_DriveRef;
   bExecute    : BOOL;
   tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
   sFirmwareVersion : STRING(20); (* version string vx_yy_bnnnn, e.g. "v1.05_b0009" for v1.05 Build 0009*)
   sFirmwarePath : T_MaxString; (* drive:\path, e.g. "C:\TwinCAT\Io\TcDriveManager\FirmwarePool" *)
   sNetIdIPC : T_AmsNetId;
   iReqEcState : UINT := EC_DEVICE_STATE_OP;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stDriveRef</td>
<td>ST_DriveRef</td>
<td>Reference to the drive. The reference to the drive can be linked directly to the PLC in the System Manager. To do this, an instance of the structure ST_PlcDriveRef must be allocated and the NetID must be converted from the byte array into a string. (Type: ST_DriveRef [11])</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>The function block is enabled via a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
<tr>
<td>sFirmwareVersion</td>
<td>STRING(20)</td>
<td>Specifies the desired firmware version in the form of vx.yy_bnnnn, e.g. &quot;v1.05_b0009&quot; for Version v1.05 Build 0009.</td>
</tr>
</tbody>
</table>

**Release-Builds:**

- "v1.05_b0009" for a specific build, e.g. v1.05 Build 0009
- "v1.05_b0??" latest build of a specified version, e.g. v1.05
- "v1.??_b0??" latest build of a specified main version, e.g. v1
- "v.??_b??" latest build of the latest version

**Customer-specific Firmware-Builds:**

- "v1.05_b1009" for a specific build, e.g. v1.05 Build 0009
- "v1.05_b1??" latest build of a specified version, e.g. v1.05
- "v1.??_b1??" latest build of a specified main version, e.g. v1
- "v2.??_b10??" latest build of the latest version
- "v1.05_b8909" for a specific build, e.g. v1.05 Build 8909
- "v1.05_b8??" latest build of a specified version, e.g. v1.05
- "v1.??_b8??" latest build of a specified main version, e.g. v1
- "v2.??_b8??" latest build of the latest version

**Debug-Builds:**
**Function blocks**

### TE1000

**Version:** 1.3

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sFirmwarePath</strong></td>
<td>T_MaxString</td>
<td>Specifies the path for the firmware pool in which the firmware files are located, e.g. C:s</td>
</tr>
<tr>
<td><strong>sNetIdIPC</strong></td>
<td>T_AmsNetId</td>
<td>AMS-NetID of the controller (IPC)</td>
</tr>
<tr>
<td><strong>iReqEcState</strong></td>
<td>UINT</td>
<td>Desired EtherCAT state after the update, only if an update is actually being executed. The statuses are defined in PLC Lib Tc2_EtherCAT as global constants.</td>
</tr>
</tbody>
</table>

### Outputs

**VAR_OUTPUT**

- **bBusy**: BOOL
- **bError**: BOOL
- **iAdsErrId**: UINT
- **iSercosErrId**: UINT
- **iDiagNumber**: UDINT
- **eFwUpdateState**: E_FwUpdateState
- **iLoadProgress**: INT
- **sSelectedFirmwareFile**: STRING(MAX_STRING_LENGTH); (* found firmware file, e.g. "AX5yxx_xxxx_0010 v1_05_b0009.efw" *)

### Sample

**VAR CONSTANT**

- **iNumOfDrives**: INT := 2;

**VAR**

- **bInit**: ARRAY [1..iNumOfDrives] OF BOOL := 2(TRUE);
- **fbFirmwareUpdate**: ARRAY [1..iNumOfDrives] OF FB_SoEAX5000FirmwareUpdate_ByDriveRef;
- **stPlcDriveRef**: ARRAY[1..iNumOfDrives] OF ST_PlcDriveRef;
- **stDriveRef**: ARRAY [1..iNumOfDrives] OF ST_DriveRef;
- **sFirmwareVersion**: ARRAY [1..iNumOfDrives] OF STRING(20) := 2('v1.05_b0009');
- **eFwUpdateState**: ARRAY[1..iNumOfDrives] OF E_FwUpdateState;
- **sSelectedFirmwareFile**: ARRAY [1..iNumOfDrives] OF STRING(MAX_STRING_LENGTH);
- **iUpdateState**: INT;
- **bExecute**: BOOL;
- **sNetIdIPC**: T_AmsNetId := '';
- **sFirmwarePath**: T_MaxString := 'C:s|t\inCAT\Io\TcDriveManager\FirmwarePool';

I := INT;
bAnyInit : BOOL;
bAnyBusy : BOOL;
bAnyError : BOOL;
END_VAR

CASE iUpdateState OF
  0:
    bAnyInit := FALSE;
    FOR I := 1 TO iNumOfDrives DO
      IF bInit[I] THEN
        bAnyInit := TRUE;
        stDriveRef[I].sNetId := F_CreateAmsNetId(stPlcDriveRef[I].aNetId);
        stDriveRef[I].nSlaveAddr := stPlcDriveRef[I].nSlaveAddr;
        stDriveRef[I].nDriveNo := stPlcDriveRef[I].nDriveNo;
        stDriveRef[I].nDriveType := stPlcDriveRef[I].nDriveType;
        IF (stDriveRef[I].sNetId <> '') AND (stDriveRef[I].nSlaveAddr <> 0)
          THEN bInit[I] := FALSE;
        END_IF
      END_IF
    END_FOR
    IF NOT bAnyInit AND bExecute THEN
      iUpdateState := 1;
    END_IF
  1:
    FOR I := 1 TO iNumOfDrives DO
      fbFirmwareUpdate[I](
        stDriveRef := stDriveRef[I],
        bExecute := TRUE,
        tTimeout := T#15s,
        sFirmwareVersion := sFirmwareVersion[I],
        sFirmwarePath := sFirmwarePath,
        sNetIdIPC := sNetIdIPC,
        iReqEcState := EC_DEVICE_STATE_OP,
        eFwUpdateState =>$ eFwUpdateState[I],
      );
    END_FOR
    iUpdateState := 2;
  2:
    bAnyBusy := FALSE;
    bAnyError := FALSE;
    FOR I := 1 TO iNumOfDrives DO
      fbFirmwareUpdate[I](
        eFwUpdateState =>$ eFwUpdateState[I],
        sSelectedFirmwareFile =>$ sSelectedFirmwareFile[I],
      );
      IF NOT fbFirmwareUpdate[I].bBusy THEN
        fbFirmwareUpdate[I](bExecute := FALSE);
        IF fbFirmwareUpdate[I].bError THEN
          bAnyError := TRUE;
        END_IF
        ELSE
          bAnyBusy := TRUE;
        END_IF
      END_FOR
      IF NOT bAnyBusy THEN
        bExecute := FALSE;
        IF NOT bAnyError THEN
          iUpdateState := 0; (* OK *)
        ELSE
          iUpdateState := 0; (* Error *)
        END_IF
      END_IF
    END_CASE
4.3 IndraDrive Cs

4.3.1 Conversion functions

4.3.1.1 F_ConvWordToSTIndraDriveCsC1D

With this function the Class 1 diagnosis \( \text{FB}_\text{SoEReadClassXDiag}_\text{ByDriveRef} [\rightarrow 24] \) (S-0-0011) can be changed to an \( \text{ST}_\text{IndraDriveCs}_\text{C1D} [\rightarrow 50] \) structure.

**Inputs**

VAR_INPUT

\[ \text{wClass1Diag : WORD}; \]

**Name** | **Type** | **Description**
--- | --- | ---
\[ \text{wClass1Diag} \] | WORD | Class 1 diagnosis Word from S-0-0011 (see \( \text{FB}_\text{SoEReadClassXDiag}_\text{ByDriveRef} [\rightarrow 24]) \).

**Return value**

FUNCTION \( \text{F_ConvWordToSTIndraDriveCsC1D} : \text{ST}_\text{IndraDriveCs}_\text{C1D} \)

**Name** | **Type** | **Description**
--- | --- | ---
\[ \text{F_ConvWordToSTIndraDriveCsC1D} \] | \( \text{ST}_\text{IndraDriveCs}_\text{C1D} [\rightarrow 50] \) | Return value of the function Class-1-Diagnosis as \( \text{ST}_\text{IndraDriveCs}_\text{C1D} \) structure.

4.3.1.2 F_ConvWordToSTIndraDriveCsDriveStatus

With this function the Drive status word (S-0-0135) can be changed to a structure \( \text{ST}_\text{IndraDriveCsDriveStatus} [\rightarrow 50] \).

**Inputs**

VAR_INPUT

\[ \text{wClass1Diag : WORD}; \]

**Name** | **Type** | **Description**
--- | --- | ---
\[ \text{wClass1Diag} \] | WORD | Drive status word from S-0-0135 Can be read with \( \text{FB}_\text{SoE}_\text{Read}_\text{ByDriveRef} \), can be mapped if necessary.

**Return value**

FUNCTION \( \text{F_ConvWordToSTIndraDriveCsDriveStatus} : \text{ST}_\text{IndraDriveCsDriveStatus} \)
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ConvWordToST_IndraDriveCsDriveStatus</td>
<td>ST_IndraDriveCsDriveStatus [50]</td>
<td>Return value of the function. Drive status word as ST_IndraDriveCsDriveStatus structure.</td>
</tr>
</tbody>
</table>

### 4.4 F_GetVersionTcDrive

This function can be used to read PLC library version information.

**Inputs**

```plaintext
VAR_INPUT
  nVersionElement : INT;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nVersionElement</td>
<td>INT</td>
<td>nVersionElement: version element to be read. Possible parameters: • 1: major number; • 2: minor number; • 3: revision number;</td>
</tr>
</tbody>
</table>

**Return value**

```plaintext
FUNCTION F_GetVersionTcDrive : UINT
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_GetVersionTcDrive</td>
<td>UINT</td>
<td>Return value of the function. Version element as UINT.</td>
</tr>
</tbody>
</table>

### 4.5 SimplePlcMotion

Simple PLC Motion function blocks enable easy operation of a drive directly from the PLC.

#### 4.5.1 FB_CoEDriveEnable

The FB_CoEDriveEnable function block enables a CoE drive in order to be able to subsequently supply it with setpoints using the function block FB_CoEDriveMoveVelocity [41].
## Inputs

```plaintext
VAR_INPUT
  bEnable : BOOL;
  bReset  : BOOL;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
<td>Activates the CoE drive.</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL</td>
<td>Performs a drive reset in the event of a fault. &quot;Bit 7&quot; is set in the drive control word.</td>
</tr>
</tbody>
</table>

## / Inputs/Outputs

```plaintext
VAR_IN_OUT
  stCoeDriveIoInterface : ST_CoeDriveIoInterface;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCoeDriveIoInterface</td>
<td>ST_CoeDriveIoInterface</td>
<td>Data structure to which the process image of the CoE drive must be linked.</td>
</tr>
</tbody>
</table>

## Outputs

```plaintext
VAR_OUTPUT
  bStatus     : BOOL;
  bDriveError : BOOL;
END_VAR
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStatus</td>
<td>BOOL</td>
<td>If bStatus=TRUE, the drive is ready for operation and follows the setpoints.</td>
</tr>
<tr>
<td>bDriveError</td>
<td>BOOL</td>
<td>The drive is in the error state.</td>
</tr>
</tbody>
</table>

## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1.4024.22</td>
<td>PC or CX (x86 or x64)</td>
<td>Tc2_Drive</td>
</tr>
</tbody>
</table>

### 4.5.2 FB_CoEDriveMoveVelocity

The **FB_CoEDriveMoveVelocity** function block generates a simple three-phase velocity profile (without jerk limitation) that can be used to supply a CoE drive directly. Different accelerations or decelerations can be used below and above a parameterizable velocity threshold. The target velocity can be changed during operation.

The CoE drive must first be enabled via the function block **FB_CoEDriveEnable** [p. 40].
## Inputs

### VAR_INPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
<td>Activates the setpoint generation.</td>
</tr>
<tr>
<td>fVelocity</td>
<td>LREAL</td>
<td>Target velocity. fVelocity can be changed during operation.</td>
</tr>
<tr>
<td>fAcceleration1</td>
<td>LREAL</td>
<td>Acceleration 1 is used below the parameterized velocity threshold stOptions.</td>
</tr>
<tr>
<td>fAcceleration2</td>
<td>LREAL</td>
<td>Acceleration 2 is used above the parameterized velocity threshold stOptions.</td>
</tr>
<tr>
<td>fDeceleration1</td>
<td>LREAL</td>
<td>Deceleration 1 is used below the parameterized velocity threshold stOptions.</td>
</tr>
<tr>
<td>fDeceleration2</td>
<td>LREAL</td>
<td>Deceleration 2 is used above the parameterized velocity threshold stOptions.</td>
</tr>
<tr>
<td>bNegativeDirection</td>
<td>BOOL</td>
<td>bNegativeDirection reverses the direction of travel.</td>
</tr>
<tr>
<td>stOptions</td>
<td>ST_DriveMoveVelocityOptions</td>
<td>Data structure with additional parameters.</td>
</tr>
</tbody>
</table>

### Inputs/Outputs

### VAR_IN_OUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCoEDriveIoInterface</td>
<td>ST_CoeDriveIoInterface</td>
<td>Process image of the CoE drive</td>
</tr>
</tbody>
</table>

## Outputs

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bInVelocity</td>
<td>BOOL</td>
<td>Target velocity is reached.</td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>bBusy is TRUE as long as the function block is active and a setpoint profile is being calculated.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>bError becomes TRUE if an error occurs.</td>
</tr>
<tr>
<td>iErrorID</td>
<td>UDINT</td>
<td>Error number</td>
</tr>
<tr>
<td>fActualVelocity</td>
<td>LREAL</td>
<td>Currently reached velocity of the drive.</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1.4024.22</td>
<td>PC or CX (x86 or x64)</td>
<td>Tc2_Drive</td>
</tr>
</tbody>
</table>

### 4.5.3 FB_SoEDriveEnable

The FB_SoEDriveEnable function block enables a SoE drive in order to be able to subsequently supply it with setpoints using the function block FB_SoEDriveMoveVelocity [p. 44].

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
<td>Activates the SoE drive.</td>
</tr>
</tbody>
</table>

#### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSoeDriveIoInterface</td>
<td>ST_SoEDriveIoInterface</td>
<td>Data structure to which the process image of the SoE drive must be linked.</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStatus</td>
<td>BOOL</td>
<td>If bStatus=TRUE, the drive is ready for operation and follows the setpoints.</td>
</tr>
<tr>
<td>bDriveError</td>
<td>BOOL</td>
<td>The drive is in the error state.</td>
</tr>
</tbody>
</table>

Requirements
4.5.4 FB_SoEDriveMoveVelocity

The FB_SoEDriveMoveVelocity function block generates a simple three-phase velocity profile (without jerk limitation) that can be used to supply an SoE drive directly. Different accelerations or decelerations can be used below and above a parameterizable velocity threshold. The target velocity can be changed during operation.

The SoE drive must first be enabled via the function block FB_SoEDriveEnable [43].

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
<td>Activates the setpoint generation.</td>
</tr>
<tr>
<td>fVelocity</td>
<td>LREAL</td>
<td>Target velocity. fVelocity can be changed during operation.</td>
</tr>
<tr>
<td>fAcceleration1</td>
<td>LREAL</td>
<td>Acceleration 1 is used below the parameterized velocity threshold stOptions. fVelocityThreshold.</td>
</tr>
<tr>
<td>fAcceleration2</td>
<td>LREAL</td>
<td>Acceleration 2 is used above the parameterized velocity threshold stOptions. fVelocityThreshold.</td>
</tr>
<tr>
<td>fDeceleration1</td>
<td>LREAL</td>
<td>Deceleration 1 is used below the parameterized velocity threshold stOptions. fVelocityThreshold.</td>
</tr>
<tr>
<td>fDeceleration2</td>
<td>LREAL</td>
<td>Deceleration 2 is used above the parameterized velocity threshold stOptions. fVelocityThreshold.</td>
</tr>
<tr>
<td>bNegativeDirection</td>
<td>BOOL</td>
<td>bNegativeDirection reverses the direction of travel.</td>
</tr>
<tr>
<td>stOptions</td>
<td>ST_DriveMoveVelocityOptions</td>
<td>Data structure with additional parameters.</td>
</tr>
</tbody>
</table>

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSoEDriveIoInterface</td>
<td>ST_SoeDriveIoInterface</td>
<td>Process image of the CoE drive</td>
</tr>
</tbody>
</table>
## Outputs

**VAR_OUTPUT**

- `bInVelocity` : BOOL;  
- `bBusy` : BOOL;  
- `bError` : BOOL;  
- `iErrorID` : UDINT;  
- `fActualVelocity` : LREAL;

### Name | Type | Description
---|---|---
`bInVelocity` | BOOL | Target velocity is reached.
`bBusy` | BOOL | `bBusy` is TRUE as long as the function block is active and a setpoint profile is being calculated.
`bError` | BOOL | `bError` becomes TRUE if an error occurs.
`iErrorID` | UDINT | Error number
`fActualVelocity` | LREAL | Currently reached velocity of the drive.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1.4024.22</td>
<td>PC or CX (x86 or x64)</td>
<td>Tc2_Drive</td>
</tr>
</tbody>
</table>
5 Data types

5.1 General SoE

5.1.1 ST_SoE_String

The ST_SoE_String structure describes a string as it can be used in SoE accesses.

```c
TYPE ST_SoE_String :
  STRUCT
    iActualSize : UINT;
    iMaxSize : UINT;
    strData : STRING(MAX_STRING_LENGTH);
  END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iActualSize</td>
<td>UINT</td>
<td>Current length of the string (without closing \0)</td>
</tr>
<tr>
<td>iMaxSize</td>
<td>UINT</td>
<td>Maximum length of the string (without closing \0)</td>
</tr>
<tr>
<td>strData</td>
<td>STRING(MAX_STRING_LENGTH)</td>
<td>String</td>
</tr>
</tbody>
</table>

5.1.2 ST_SoE_StringEx

The structure ST_SoE_StringEx describes a string such as can be used with SoE accesses, including the preset parameter attribute.

```c
TYPE ST_SoE_StringEx :
  STRUCT
    dwAttribute : DWORD;
    iActualSize : UINT;
    iMaxSize : UINT;
    strData : STRING(MAX_STRING_LENGTH);
  END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwAttribute</td>
<td>DWORD</td>
<td>Parameter attribute</td>
</tr>
<tr>
<td>iActualSize</td>
<td>UINT</td>
<td>Current length of the string (without closing \0)</td>
</tr>
<tr>
<td>iMaxSize</td>
<td>UINT</td>
<td>Maximum length of the string (without closing \0)</td>
</tr>
<tr>
<td>strData</td>
<td>STRING(MAX_STRING_LENGTH)</td>
<td>String</td>
</tr>
</tbody>
</table>

5.1.3 List types

5.1.3.1 ST_SoE_DiagNumList

The ST_SoE_DiagNumList structure contains the list length (Minimum, Maximum) in bytes as well as the history of the diagnosis numbers.

```c
TYPE ST_SoE_DiagNumList :
  STRUCT
    iActualSize : UINT;
    iMaxSize : UINT;
  END_STRUCT
END_TYPE
```
Data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iActualSize</td>
<td>UINT</td>
<td>Current length of the string (without closing 0)</td>
</tr>
<tr>
<td>iMaxSize</td>
<td>UINT</td>
<td>Maximum length of the string (without closing 0)</td>
</tr>
<tr>
<td>arrDiagNumbers</td>
<td>ARRAY [0..29] OF UDINT</td>
<td>List of the maximum 30 last error numbers (as UDINT)</td>
</tr>
</tbody>
</table>

5.2 AX5000 SoE

5.2.1 E_FwUpdateState

The enumeration E_FwUpdateState describes the state of a firmware update.

```
TYPE E_SoE_CmdState := (
  (* update states *)
  eFwU_NoError := 0,
  eFwU_CheckCfgIdentity,
  eFwU_CheckSlaveCount,
  eFwU_CheckFindSlavePos,
  eFwU_WaitForScan,
  eFwU_ScanningSlaves,
  eFwU_CheckScannedIdentity,
  eFwU_CheckScannedFirmware,
  eFwU_FindFirmwareFile,
  eFwU_WaitForUpdate,
  eFwU_WaitForSlaveState,
  eFwU_StartFwUpdate,
  eFwU_FwUpdateInProgress,
  eFwU_FwUpdateDone,
  eFwU_NoFwUpdateRequired,

  (* not updating via this channel *)
  eFwU_UpdateViaOtherChannelActive,
  eFwU_UpdatedViaOtherChannel,

  (* error states *)
  eFwU_GetSlaveIdentityError := -1,
  eFwU_GetSlaveCountError := -2,
  eFwU_GetSlaveAddrError := -3,
  eFwU_StartScanError := -4,
  eFwU_ScanStateError := -5,
  eFwU_ScanIdentityError := -6,
  eFwU_GetSlaveStateError := -7,
  eFwU_ScanFirmwareError := -8,
  eFwU_FindFileError := -9,
  eFwU_CfgTypeInNoAX5xxx := -10,
  eFwU_ScannedTypeInNoAX5xxx := -11,
  eFwU_ChannelMismatch := -12,
  eFwU_ChannelMismatch_1Cfg_2Scanned := -13,
  eFwU_ChannelMismatch_2Cfg_1Scanned := -14,
  eFwU_CurrentMismatch := -15,
  eFwU_FwUpdateError := -16,
  eFwU_ReqSlaveStateError := -17
);
END_TYPE
```

Update Status

<table>
<thead>
<tr>
<th>eFwU_NoError</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>eFwU_CheckCfgIdentity</td>
<td>Read the configured slave types (number of channels, current, revision)</td>
</tr>
<tr>
<td>eFwU_CheckSlaveCount</td>
<td>Determine the configured number of slaves</td>
</tr>
<tr>
<td>eFwU_CheckFindSlavePos</td>
<td>Search for the slave address in the master object directory</td>
</tr>
<tr>
<td>eFwU_WaitForScan</td>
<td>Wait for online scan</td>
</tr>
</tbody>
</table>
### Update error

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eFwU_GetSlaveIdentityError</td>
<td>Reading of the configured slave type failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_GetSlaveCountError</td>
<td>Determination of the configured number of slaves failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_GetSlaveAddrError</td>
<td>Search for the slave address in the master object directory failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_StartScanError</td>
<td>Start of the online scan failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_ScanStateError</td>
<td>Online scan failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_ScanIdentityError</td>
<td>Reading of the scanned slave type (number of channels, current, revision) failed (see iAdsErrId)-</td>
</tr>
<tr>
<td>eFwU_GetSlaveStateError</td>
<td>Determination of the EtherCAT slave status failed (see iAdsErrld)-</td>
</tr>
<tr>
<td>eFwU_ScanFirmwareError</td>
<td>Reading of the firmware version failed (see iAdsErrld + iSercosErrId).</td>
</tr>
<tr>
<td>eFwU_FindFileError</td>
<td>Search for the selected firmware file failed (see iAdsErrld).</td>
</tr>
<tr>
<td>eFwU_CfgTypeInNoAX5xxx</td>
<td>The configured slave is not an AX5000.</td>
</tr>
<tr>
<td>eFwU_ScannedTypeInNoAX5xxx</td>
<td>The scanned slave is not an AX5000.</td>
</tr>
<tr>
<td>eFwU_ChannelMismatch</td>
<td>The number of configured and found channels of the AX5000 do not match.</td>
</tr>
<tr>
<td>eFwU_ChannelMismatch_1Cfg_2Scanned</td>
<td>Single-channel device configured, but two-channel device found.</td>
</tr>
<tr>
<td>eFwU_ChannelMismatch_2Cfg_1Scanned</td>
<td>Two-channel device configured, but single-channel device found.</td>
</tr>
<tr>
<td>eFwU_CurrentMismatch</td>
<td>AX5000 type does not match in terms of current, e.g. AX5103 (3 A) configured, but AX5106 (6 A) found.</td>
</tr>
<tr>
<td>eFwU_FwUpdateError</td>
<td>General update error (see iAdsErrId)</td>
</tr>
<tr>
<td>eFwU_ReqSlaveStateError</td>
<td>Switching to the desired EtherCAT status failed.</td>
</tr>
</tbody>
</table>

### 5.2.2 ST_AX5000_C1D for Class 1 diagnosis

```plaintext
TYPE ST_AX5000_C1D:
  STRUCT
    bOverloadShutdown : BOOL; (* CID Bit 0 * )
    bAmplifierOverTempShutdown : BOOL; (* CID Bit 1 * )
    bMotorOverTempShutdown : BOOL; (* CID Bit 2 * )
    bCoolingErrorShutdown : BOOL; (* CID Bit 3 * )
    bControlVoltageError : BOOL; (* CID Bit 4 * )
    bFeedbackError : BOOL; (* CID Bit 5 * )
    bCommunicationError : BOOL; (* CID Bit 6 * )
    bOverCurrentError : BOOL; (* CID Bit 7 * )
    bOverVoltageError : BOOL; (* CID Bit 8 * )
    bUnderVoltageError : BOOL; (* CID Bit 9 * )
    bPowerSupplyPhaseError : BOOL; (* CID Bit 10 * )
    bExcessivePosDiviationError : BOOL; (* CID Bit 11 * )
```

5.2.3 ST_AX5000DriveStatus

TYPE ST_AX5000DriveStatus :
STRUCT
  bStatusCmdValProcessing : BOOL;
  bRealTimeStatusBit1     : BOOL;
  bRealTimeStatusBit2     : BOOL;
  bDrvShutdownBitC1D      : BOOL;
  bChangeBitC2D           : BOOL;
  bChangeBitC3D           : BOOL;
  bNotReadyToPowerUp      : BOOL;
  bReadyForPower          : BOOL;
  bReadyForEnable         : BOOL;
  bEnabled                : BOOL;
  iActOpModeParNum        : UINT;
  eActOpMode              : E_AX5000_DriveOpMode;
  iReserved               : UINT;
END_STRUCT
END_TYPE

5.2.4 E_AX5000_DriveOpMode

TYPE E_AX5000_DriveOpMode : ( eOPM_NoModeOfOperation := 0, eOPM_TorqueCtrl     := 1, eOPM_VeloCtrl      := 2, eOPM_PosCtrlFbk1   := 3, eOPM_PosCtrlFbk2   := 4, eOPM_PosCtrlFbk1LagLess := 11, eOPM_PosCtrlFbk2LagLess := 12 );
END_TYPE

5.3 IndraDrive Cs

5.3.1 E_IndraDriveCs_DriveOpMode

5.3.2 ST_IndraDriveCs_C1D for Class 1 diagnosis

TYPE ST_IndraDriveCs_C1D : STRUCT
  bOverloadShutdown : BOOL; (* CID Bit 0 *)
  bAmplifierOverTempShutdown : BOOL; (* CID Bit 1 *)
  bMotorOverTempShutdown : BOOL; (* CID Bit 2 *)
  bReserved_3 : BOOL; (* CID Bit 3 *)
  bControlVoltageError : BOOL; (* CID Bit 4 *)
  bFeedbackError : BOOL; (* CID Bit 5 *)
  bReserved_6 : BOOL; (* CID Bit 6 *)
  bOverCurrentError : BOOL; (* CID Bit 7 *)
  bOverVoltageError : BOOL; (* CID Bit 8 *)
  bUnderVoltageError : BOOL; (* CID Bit 9 *)
  bReserved_10 : BOOL; (* CID Bit 10 *)
  bExcessivePosDiviationError : BOOL; (* CID Bit 11 *)
  bCommunicationErrorBit : BOOL; (* CID Bit 12 *)
  bOvertravelLimitExceeded : BOOL; (* CID Bit 13 *)
  bReserved_14 : BOOL; (* CID Bit 14 *)
  bManufacturerSpecificError : BOOL; (* CID Bit 15 *)
END_STRUCT
END_TYPE

5.3.3 ST_IndraDriveCsDriveStatus

TYPE ST_IndraDriveCsDriveStatus : STRUCT
  bStatusCmdValProcessing : BOOL;
  bRealTimeStatusBit1 : BOOL;
}
### 5.4 SERCOS

#### 5.4.1 E_SoE_AttribLen

The enumeration `E_SoE_AttribLen` in the attribute of a parameter specifies whether the value of the parameter is a 2, 4, or 8-byte data type (single value), or whether it is a list consisting of 1, 2, 4, or 8-byte data types. List types (with `eSoE_LEN_V...`) first have the current list length in bytes (in a 16 bit value), then the maximum list length in bytes (in a 16 bit value) and the actual list in the specified data type.

**Sample:** see `ST_SoE_String [46]` of the `eSoE_LEN_V1BYTE` type.

```
TYPE E_SoE_AttribLen : (
  eSoE_LEN_2BYTE  := 1,
  eSoE_LEN_4BYTE  := 2,
  eSoE_LEN_8BYTE  := 3,
  eSoE_LEN_V1BYTE := 4,
  eSoE_LEN_V2BYTE := 5,
  eSoE_LEN_V4BYTE := 6,
  eSoE_LEN_V8BYTE := 7
);
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eSoE_LEN_2BYTE</code></td>
<td>2-byte data type (e.g. UINT, INT, WORD, IDN)</td>
</tr>
<tr>
<td><code>eSoE_LEN_4BYTE</code></td>
<td>4-byte data type (e.g. UDINT, DINT, DWORD, REAL)</td>
</tr>
<tr>
<td><code>eSoE_LEN_8BYTE</code></td>
<td>8-byte data type (e.g. ULINT, LINT, LREAL)</td>
</tr>
<tr>
<td><code>eSoE_LEN_V1BYTE</code></td>
<td>List of 1-byte data types (e.g. string)</td>
</tr>
<tr>
<td><code>eSoE_LEN_V2BYTE</code></td>
<td>List of 2-byte data types (e.g. IDN list)</td>
</tr>
<tr>
<td><code>eSoE_LEN_V4BYTE</code></td>
<td>List of 4-Byte data types</td>
</tr>
<tr>
<td><code>eSoE_LEN_V8BYTE</code></td>
<td>List of 8-Byte data types</td>
</tr>
</tbody>
</table>

#### 5.4.2 E_SoE_CmdControl

The enumeration `E_SoECmdControl` determines whether the command is to be canceled, set or started.

```
TYPE E_SoE_CmdControl :
  eSoE_CmdControl_Cancel := 0,
  eSoE_CmdControl_Set   := 1,
  eSoE_CmdControl_SetAndEnable := 3
);
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eSoE_CmdControl_Cancel</code></td>
<td>Cancel command.</td>
</tr>
<tr>
<td><code>eSoE_CmdControl_Set</code></td>
<td>Set command.</td>
</tr>
<tr>
<td><code>eSoE_CmdControl_SetAndEnable</code></td>
<td>Set command and execute.</td>
</tr>
</tbody>
</table>
5.4.3 E_SoE_CmdState

The enumeration E_SoE_CmdState describes the state of an SoE command.

```plaintext
TYPE E_SoE_CmdState : (  
  eSoE_CmdState_NotSet := 0,  
  eSoE_CmdState_Set := 1,  
  eSoE_CmdState_Executed := 2,  
  eSoE_CmdState_SetEnabledExecuted := 3,  
  eSoE_CmdState_SetAndInterrupted := 5,  
  eSoE_CmdState_SetEnabledNotExecuted := 7,  
  eSoE_CmdState_Error := 15  
);  
END_TYPE  
```

eSoE_CmdState_NotSet = 0
- kein Kommando aktiv

eSoE_CmdState_Set = 1
- Kommando gesetzt (vorbereitet) aber (noch) nicht ausgeführt

eSoE_CmdState_Executed = 2
- Kommando wurde ausgeführt

eSoE_CmdState_SetEnabledExecuted = 3
- Kommando gesetzt (vorbereitet) und ausgeführt

eSoE_CmdState_SetAndInterrupted = 5
- Kommando wurde gesetzt aber unterbrochen

eSoE_CmdState_SetEnabledNotExecuted = 7
- Kommandoausführung ist noch aktiv

eSoE_CmdState_Error = 15
- Fehler bei der Kommandoausführung, es wurde in den Fehlerstate gewechselt

5.4.4 E_SoE_Type

The enumeration E_SoE_Type describes the representation of the parameter value in the attribute of the parameter.

```plaintext
TYPE E_SoE_Type : (  
  eSoE_Type_BIN := 0,  
  eSoE_Type_UNSIGNED := 1,  
  eSoE_Type_SIGNED := 2,  
  eSoE_Type_HEX := 3,  
  eSoE_Type_TEXT := 4,  
  eSoE_Type_IDN := 5,  
  eSoE_Type_FLOAT := 6  
);  
END_TYPE  
```

The enumeration E_SoE_Type is used to determine how the data can be interpreted:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSoE_Type_BIN</td>
<td>Binary</td>
</tr>
<tr>
<td>eSoE_Type_UNSIGNED</td>
<td>Integer without sign</td>
</tr>
<tr>
<td>eSoE_Type_SIGNED</td>
<td>Integer with sign</td>
</tr>
<tr>
<td>eSoE_Type_HEX</td>
<td>Hexadecimal number</td>
</tr>
<tr>
<td>eSoE_Type_TEXT</td>
<td>Text</td>
</tr>
<tr>
<td>eSoE_Type_IDN</td>
<td>Parameter number</td>
</tr>
<tr>
<td>eSoE_Type_FLOAT</td>
<td>Floating point number</td>
</tr>
</tbody>
</table>
5.5 SimplePlcMotion

5.5.1 E_CoEDriveEnableState

```
TYPE E_CoEDriveEnableState : (  
eCoEDriveEnableState_ReadyToSwitchOn := 0,  
eCoEDriveEnableState_SwitchedOn := 1,  
eCoEDriveEnableState_OperationEnabled := 2,  
eCoEDriveEnableState_Fault := 3,  
eCoEDriveEnableState_FaultReactionActive := 4,  
eCoEDriveEnableState_NotReadySwitchedOn := 5,  
eCoEDriveEnableState_SwitchedOnDisabled := 6,  
eCoEDriveEnableState_DriveFollows := 7  
);  
END_TYPE
```

5.5.2 E_DriveMoveVelocityError

```
TYPE E_CoEDriveEnableState : (  
  DRIVEPLCERROR_DRIVENOTREADY := 16#4BF0,  
  DRIVEPLCERROR_INVALIDDYNAMICSPARAMETER,  
  DRIVEPLCERROR_INVALIDVELOCITYSCALING,  
  DRIVEPLCERROR_FUNCTIONBLOCKSUIDDLYDISABLED 
);  
END_TYPE
```

5.5.3 ST_CoEDriveIoInterface

Data structure for mapping the process image of a CoE drive to use the function blocks FB_CoEDriveEnable [40] and FB_CoEDriveMoveVelocity [41].

```
TYPE ST_CoEDriveIoInterface :  
  STRUCT  
    iControl : UINT;  
    iStatus : UINT;  
    iCmdVelo : DINT;  
    iActVelo : DINT;  
    stAdsAddr : ST_AmsAddr;  
    iChannel : BYTE;  
    eStateMachine : E_CoEDriveEnableState;  
  END_STRUCT  
END_TYPE
```

5.5.4 ST_DriveMoveVelocityOptions

The structure ST_DriveMoveVelocityOptions describes additional parameters of the function blocks FB_CoEDriveMoveVelocity [41] and FB_SoEDriveMoveVelocity [44].

```
TYPE ST_DriveMoveVelocityOptions :  
  STRUCT  
    bVelocityUnitRPM : BOOL;  
    fVelocityThreshold : LREAL;  
    fVelocityScalingFactor : LREAL;  
    fFilterTimeActualVelocity : LREAL;  
    bInvertDirection : BOOL;  
  END_STRUCT  
END_TYPE
```

5.5.5 ST_SoEDriveIoInterface

Data structure for mapping the process image of a SoE drive to use the function blocks FB_SoEDriveEnable [43] and FB_SoEDriveMoveVelocity [44].
TYPE ST_SoEDriveIoInterface :
  STRUCT
    iMasterControlWord   : WORD;
    iVelocityCommandValue : DINT;
    iDriveStatusWord     : WORD;
    iVelocityFeedbackValue : DINT;
    iState               : UINT;
    stAdsAddr            : ST_AmsAddr;
    iChannel             : BYTE;
  END_STRUCT
END_TYPE
6 Samples

Sample project and sample configuration for AX5000 diagnostics
Download: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_Drive/Resources/zip/2307584011.zip

Sample project and sample configuration for IndraDrive Cs diagnostics
Download: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_Drive/Resources/zip/2307586955.zip
More Information:
www.beckhoff.com/te1000