# Table of Contents

1 Foreword ................................................................................................................................. 7
   1.1 Notes on the documentation ............................................................................................. 7
   1.2 Safety instructions ............................................................................................................ 8
   1.3 Notes on information security .......................................................................................... 9

2 Overview ................................................................................................................................... 10

3 EtherCAT Commands .................................................................................................................. 11
   3.1 FB_EcPhysicalReadCmd ................................................................................................. 11
   3.2 FB_EcPhysicalWriteCmd ............................................................................................... 12
   3.3 FB_EcLogicalReadCmd ................................................................................................. 14
   3.4 FB_EcLogicalWriteCmd ............................................................................................... 15

4 EtherCAT Diagnostic ................................................................................................................... 17
   4.1 FB_EcGetAllSlaveAbnormalStateChanges ...................................................................... 17
   4.2 FB_EcGetAllSlaveAddr ................................................................................................. 18
   4.3 FB_EcGetAllCrcErrors ................................................................................................. 19
   4.4 FB_EcGetAllSlavePresentStateChanges ........................................................................ 20
   4.5 FB_EcGetConfSlaves .................................................................................................... 21
   4.6 FB_EcGetLastProtErrInfo ............................................................................................. 22
   4.7 FB_EcGetMasterDevState ............................................................................................. 23
   4.8 FB_EcGetScannedSlaves ............................................................................................... 24
   4.9 FB_EcGetSlaveCount .................................................................................................... 25
   4.10 FB_EcGetSlaveCrcError .............................................................................................. 26
   4.11 FB_EcGetSlaveCrcErrorEx ......................................................................................... 27
   4.12 FB_EcGetSlaveIdentity ............................................................................................... 28
   4.13 FB_EcGetSlaveTopologyInfo ....................................................................................... 29
   4.14 FB_EcMasterFrameCount ............................................................................................. 30
   4.15 FB_EcMasterFrameStatistic .......................................................................................... 30
   4.16 FB_EcMasterFrameStatisticClearCRC ......................................................................... 31
   4.17 FB_EcMasterFrameStatisticClearFrames ....................................................................... 32
   4.18 FB_EcMasterFrameStatisticClearTxRxErr .................................................................... 33
   4.19 F_CheckVendorId .......................................................................................................... 34

5 EtherCAT State Machine .............................................................................................................. 35
   5.1 FB_EcGetAllSlaveStates ............................................................................................... 35
   5.2 FB_EcGetMasterState ................................................................................................... 36
   5.3 FB_EcGetSlaveState ..................................................................................................... 37
   5.4 FB_EcReqMasterState ................................................................................................... 38
   5.5 FB_EcReqSlaveState .................................................................................................... 39
   5.6 FB_EcSetMasterState ................................................................................................... 41
   5.7 FB_EcSetSlaveState .................................................................................................... 42

6 CoE interface ............................................................................................................................... 44
   6.1 FB_EcCoeSdoRead ......................................................................................................... 44
   6.2 FB_EcCoeSdoReadEx ..................................................................................................... 45
   6.3 FB_EcCoeSdoWrite ....................................................................................................... 46
   6.4 FB_EcCoeSdoWriteEx .................................................................................................... 48
# Table of Contents

6.5  FB_CoERead_ByDriveRef ................................................................. 49
6.6  FB_CoEWrite_ByDriveRef .............................................................. 50

7  FoE interface ..................................................................................... 53
   7.1  FB_EcFoeAccess ......................................................................... 53
   7.2  FB_EcFoeClose .................................................................. 54
   7.3  FB_EcFoeLoad ........................................................................ 54
   7.4  FB_EcFoeOpen ................................................................. 56
   7.5  FB_EcFoeReadFile ............................................................. 57

8  SoE interface ...................................................................................... 60
   8.1  FB_EcSoeRead ........................................................................ 60
   8.2  FB_EcSoeWrite .................................................................. 61
   8.3  FB_SoERead_ByDriveRef .................................................. 63
   8.4  FB_SoEWrite_ByDriveRef .................................................... 64

9  Conversion Functions ........................................................................ 66
   9.1  F_ConvBK1120CouplerStateToString .................................. 66
   9.2  F_ConvMasterDevStateToString .......................................... 66
   9.3  F_ConvProductCodeToString .............................................. 67
   9.4  F_ConvSlaveStateToString .................................................. 67
   9.5  F_ConvSlaveStateToBits ....................................................... 68
   9.6  F_ConvSlaveStateToBitsEx .................................................. 68
   9.7  F_ConvStateToString .......................................................... 68

10 Distributed Clocks .............................................................................. 70
    10.1  DCTIME32 ........................................................................... 70
     10.1.1  ConvertDcTimeToPos ..................................................... 70
     10.1.2  ConvertPosToDcTime ...................................................... 71
     10.1.3  ConvertDcTimeToPathPos .............................................. 72
     10.1.4  ConvertPathPosToDcTime ............................................ 73
    10.2  DCTIME64 ............................................................................ 74
     10.2.1  DCTIME_TO_DCTIME64 ............................................. 74
     10.2.2  DCTIME64_TO_DCTIME ........................................... 74
     10.2.3  DCTIME64_TO_DCTIMESTRUCT .................................. 74
     10.2.4  DCTIME64_TO_FILETIME64 ...................................... 75
     10.2.5  DCTIME64_TO_STRING ............................................. 75
     10.2.6  DCTIME64_TO_SYSTEMTIME .................................... 76
     10.2.7  DCTIMESTRUCT_TO_DCTIME64 .................................. 76
     10.2.8  FILETIME64_TO_DCTIME64 .................................... 77
     10.2.9  STRING_TO_DCTIME64 ........................................... 77
     10.2.10 SYSTEMTIME_TO_DCTIME64 .................................. 78
     10.2.11 FB_EcDcTimeCtrl64 ..................................................... 79
    10.3  DCTIME64 and ULLINT .......................................................... 80
     10.3.1  F_ConvExtTimeToDcTime64 ....................................... 80
     10.3.2  F_ConvTcTimeToDcTime64 .......................................... 80
     10.3.3  F_ConvTcTimeToExtTime64 ....................................... 81
     10.3.4  F_GetActualDcTime64 ............................................... 81
     10.3.5  F_GetCurDcTaskTime64 .............................................. 82
10.3.6  F_GetCurDcTickTime64 ................................................................. 82
10.3.7  F_GetCurExtTime64 ................................................................. 83
10.3.8  FB_EcExtSyncCalcTimeDiff64 .................................................. 83
10.3.9  FB_EcExtSyncCheck64 .............................................................. 84
10.4   [obsolete] ...................................................................................... 85
10.4.1  [outdated DTIME] ................................................................. 85
10.4.2  [outdated DTIME and T_LARGE_INTEGER] .................................... 92
10.4.3  DTIME64_TO_FILETIME ............................................................ 97
10.4.4  FILETIME_TO_DCTIME64 ...................................................... 98
11   [Obsolete] .......................................................................................... 99
11.1  F_GetVersionTcEtherCAT ............................................................... 99

12 Data types ......................................................................................... 100
12.1  E_EcAdressingType ................................................................. 100
12.2  E_EcFoeMode ............................................................................. 100
12.3  E_EcMbxProtType .................................................................... 100
12.4  ST_EcCrcError .......................................................................... 101
12.5  ST_EcCrcErrorEx ...................................................................... 101
12.6  ST_EcLastProtErrInfo .............................................................. 101
12.7  ST_EcMasterStatistic ............................................................... 102
12.8  ST_EcSlaveConfigData ............................................................. 102
12.9  ST_EcSlaveIdentity .................................................................. 103
12.10  ST_EcSlaveScannedData ......................................................... 103
12.11  ST_EcSlaveState ..................................................................... 104
12.12  ST_EcSlaveStateBits .............................................................. 105
12.13  ST_EcSlaveStateBitsEx .......................................................... 106
12.14  ST_PortAddr ............................................................................ 106
12.15  ST_TopologyDataEx ............................................................... 107
12.16  DCTIME_STRUCT .................................................................. 107
12.17  T_DCTIME32 .......................................................................... 108
12.18  T_DCTIME64 .......................................................................... 108
12.19  T_DCTIME ................................................................................ 109
12.20  T_HFoe ..................................................................................... 110

13 Constants ......................................................................................... 111
13.1  Global constants ....................................................................... 111
13.2  Library version ......................................................................... 112
13.3  EtherCAT mailbox protocol error codes ..................................... 113
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!

| **DANGER**     | Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons. |
| **WARNING**    | Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons. |
| **CAUTION**    | Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons. |
| **NOTE**       | Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment. |

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.
2 Overview

The PLC library Tc2_EtherCAT contains function blocks for executing services or functions on an EtherCAT master device and/or its slave devices.

Sample project and sample configuration for diagnostics

See https://infosys.beckhoff.com/content/1033/tcplclib_tc2_ethercat/Resources/zip/2364613387.zip
3 EtherCAT Commands

3.1 FB_EcPhysicalReadCmd

The function block FB_EcPhysicalReadCmd can be used to send an EtherCAT read command (FPRD, APRD, BRD) to a particular EtherCAT slave or to all EtherCAT slaves. This command can be used by the PLC to read a register or the DPRAM of the EtherCAT slave controller.

VAR_INPUT

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId;</th>
</tr>
</thead>
<tbody>
<tr>
<td>adp</td>
<td>UINT;</td>
</tr>
<tr>
<td>ado</td>
<td>UINT;</td>
</tr>
<tr>
<td>len</td>
<td>UDINT;</td>
</tr>
<tr>
<td>eType</td>
<td>E_EcAdressingType := eAdressingType_Fixed;</td>
</tr>
<tr>
<td>pDstBuf</td>
<td>PVOID;</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME := DEFAULT_ADS_TIMEOUT;</td>
</tr>
</tbody>
</table>

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

adp: This value determines which EtherCAT slave is to be addressed with this command. The meaning of this value depends on the addressing mode selected with eType:

<table>
<thead>
<tr>
<th>eType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eAdressingType_Fixed</td>
<td>The slave is addressed by means of its configured EtherCAT address. These EtherCAT addresses can be read via the function block FB_EcGetAllSlaveAddr.</td>
</tr>
<tr>
<td>eAdressingType_AutoInc</td>
<td>The slave is addressed based on its position in the ring. The first device has the address 0 (adp=0); adp is decremented by one for all subsequent slaves: 1. Slave adp = 0 2. Slave adp = 16#fff (-1) 3. Slave adp = 16#ffe(-2) 4. Slave adp = 16#ffd(-3) etc.</td>
</tr>
<tr>
<td>eAdressingType_BroadCAST</td>
<td>All slaves are addressed by this command. adp can be set to 0.</td>
</tr>
</tbody>
</table>

ado: Physical memory (DPRAM) or register to be read.

len: Number of bytes to be read.

eType: Different EtherCAT commands are sent, depending on value of eType:

<table>
<thead>
<tr>
<th>eType</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>eAdressingType_Fixed</td>
<td>Configured Address Physical Read (FPRD)</td>
</tr>
<tr>
<td>eAdressingType_AutoInc</td>
<td>Auto Increment Physical Read (APRD)</td>
</tr>
<tr>
<td>eAdressingType_BroadCAST</td>
<td>Broadcast Read (BRD)</td>
</tr>
</tbody>
</table>
The individual commands differ in terms of addressing mode (see adp).

**pDstBuf**: The address (pointer) of the receive buffer.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrId : UDINT;</td>
</tr>
<tr>
<td>wkc : UINT;</td>
</tr>
</tbody>
</table>

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**wkc**: The working counter is incremented by each EtherCAT slave that has processed this command successfully. If only one EtherCAT slave was addressed by this command, this value should therefore be 1.

### Example of an implementation in ST:

```st
PROGRAM TEST_PhysicalReadCmd
VAR
  fbReadCmd : FB_EcPhysicalReadCmd;
  bExecute : BOOL;
  value : UINT;
  adp : UINT:=16#3E9;
  ado : UINT:=16#1100;
  eType : E_EcAdressingType := eAdressingType_Fixed;
  sNetId : T_AmsNetId:='192.168.1.5.3.1';
  wkc : UINT;
END_VAR

fbReadCmd (sNetId:=sNetID, ado:=ado, adp:=adp, eType:=eType, LEN := SIZEOF(value), pDstBuf:=ADR(value), bExecute:=bExecute);
wkc := fbReadCmd.wkc;
bError:= fbReadCmd.bError;
nErrId:= fbReadCmd.nErrId;
```

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 3.2 FB_EcPhysicalWriteCmd
EtherCAT Commands

The function block `FB_EcPhysicalWriteCmd` can be used to send an EtherCAT write command (FPWR, APWR, BWR) to a particular EtherCAT slave or to all EtherCAT slaves. This command can be used by the PLC to write to a register or the DPRAM of the EtherCAT slave controller.

**VAR_INPUT**

```plaintext
VAR_INPUT
  sNetId    : T_AmsNetId;
  adp       : UINT;
  ado       : UINT;
  len       : UDINT;
  eType     : E_EcAdressingType := eAdressingType_Fixed;
  pSrcBuf   : PVOID;
  bExecute  : BOOL;
  tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **adp**: This value determines which EtherCAT slave is to be addressed with this command. The meaning of this value depends on the addressing mode selected with `eType`:

<table>
<thead>
<tr>
<th><code>eType</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eAdressingType_Fixed</td>
<td>The slave is addressed by means of its configured EtherCAT address. These EtherCAT addresses can be read via the function block <code>FB_EcGetAllSlaveAddr</code>.</td>
</tr>
<tr>
<td>eAdressingType_AutoInc</td>
<td>The slave is addressed based on its position in the ring. The first device has the address 0 (adp=0); adp is decremented by one for all subsequent slaves: 1. Slave adp = 0 2. Slave adp = 16#ffff (-1) 3. Slave adp = 16#ffe(-2) 4. Slave adp = 16#fffd(-3) etc.</td>
</tr>
<tr>
<td>eAdressingType_BroadCAST</td>
<td>All slaves are addressed by this command. adp should be set to 0.</td>
</tr>
</tbody>
</table>

- **ado**: Physical memory (DPRAM) or register to be read.
- **len**: Number of bytes to be written.
- **eType**: Different EtherCAT commands are sent, depending on the value of `eType`:

<table>
<thead>
<tr>
<th><code>eType</code></th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>eAdressingType_Fixed</td>
<td>Configured Address Physical Write (FPWR)</td>
</tr>
<tr>
<td>eAdressingType_AutoInc</td>
<td>Auto Increment Physical Write (APWR)</td>
</tr>
<tr>
<td>eAdressingType_BroadCAST</td>
<td>Broadcast Write (BWR)</td>
</tr>
</tbody>
</table>

The individual commands differ in terms of addressing mode (see `adp`).

- **pSrcBuf**: Address (pointer) of the transmit buffer.
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bBusy     : BOOL;
  bError    : BOOL;
  nErrId    : UDINT;
  wkc       : UINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
EtherCAT Commands

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**wkc**: The working counter is incremented by each EtherCAT slave that has processed this command successfully. If only one EtherCAT slave was addressed by this command, this value should therefore be 1.

**Example of an implementation in ST:**

```st
PROGRAM Test_PhysicalWriteCmd
VAR
  fbWriteCmd : FB_EcPhysicalWriteCmd;
  bExecute : BOOL;
  value : UINT := 16#5555;
  adp : UINT := 16#3E9;
  ado : UINT := 16#1100;
  eType : E_EcAdressingType := eAdressingType_Fixed;
  sNetId : T_AmsNetId := '192.168.1.5.3.1';
  wkc : UINT;
  bError : BOOL;
  nErrId : UDINT;
END_VAR

fbWriteCmd (sNetId := sNetId, ado := ado, adp := adp, eType := eType, LEN := SIZEOF(value), pSrcBuf := ADR(value), bExecute := bExecute);

wkc := fbWriteCmd.wkc;
bError := fbWriteCmd.bError;
nErrId := fbWriteCmd.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 3.3 **FB_EcLogicalReadCmd**

The master sends a logical EtherCAT read command (LRD) with the function block FB_EcLogicalReadCmd. In each slave, local address ranges (DPRAM) can be mapped to global logical address ranges. This command therefore addresses all EtherCAT slaves, which have mapping configured for the selected logical address range.

**VAR_INPUT**

```st
VAR_INPUT
  sNetId : T_AmsNetId;
  logAddr : UDINT;
  len : UDINT;
  pDstBuf : PVOID;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**logAddr**: Logical address.

**len**: Number of bytes to be read.
**EtherCAT Commands**

**pDstBuf**: Address (pointer) to the receive buffer.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrId : UDINT;</td>
</tr>
<tr>
<td>wkc : UINT;</td>
</tr>
</tbody>
</table>

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**wkc**: The working counter is incremented by each EtherCAT slave that has processed this command successfully. If only one EtherCAT slave was addressed by this command, this value should therefore be 1.

**Example of an implementation in ST:**

```st
PROGRAM Test_LogicalReadCmd
VAR
   fbReadCmd : FB_EcLogicalReadCmd;
   bExecute : BOOL;
   value : USINT;
   logAddr : UDINT :=16#10000;
   sNetId : T_AmsNetId:='192.168.1.5.3.1';
   wkc : UINT;
   bError : BOOL;
   nErrId : UDINT;
END_VAR

   fbReadCmd (sNetId:=sNetID, logAddr:=logAddr, LEN := SIZEOF(value), pDstBuf:=ADR(value), bExecute:=bExecute);
   wkc := fbReadCmd.wkc;
   bError:= fbReadCmd.bError;
   nErrId:= fbReadCmd.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 3.4 FB_EcLogicalWriteCmd

The master sends a logical EtherCAT write command (LWR) with the function block `FB_EcLogicalWriteCmd`. In each slave, local address ranges (DPRAM) can be mapped to global logical address ranges. This command therefore addresses all EtherCAT slaves, which have mapping configured for the selected logical address range.
VAR_INPUT
VAR_INPUT
sNetId : T_AmsNetId;
logAddr : UDINT;
len : UDINT;
pSrcBuf : PVOID;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

logAddr: Logical address.

len: Number of bytes to be written.

pSrcBuf: Address (pointer) to the send buffer.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT
VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
wkc : UINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

wkc: The working counter is incremented by each EtherCAT slave that has processed this command successfully. If only one EtherCAT slave was addressed by this command, this value should therefore be 1.

Example of an implementation in ST:

PROGRAM Test_LogicalWriteCmd
VAR
  fbWriteCmd : FB_EcLogicalWriteCmd;
bExecute : BOOL;
value : USINT :=16#55;
logAddr : UDINT :=16#10000;
sNetId := T_AmsNetId:='192.168.1.5.3.1';
wkc := UINT;
bError : BOOL;
nErrId : UDINT;
END_VAR

fbWriteCmd (sNetId:=sNetId, logAddr:=logAddr, LEN := SIZEOF(value), pSrcBuf:=ADR(value), bExecute:=bExecute);
wkc := fbWriteCmd.wkc;
bError :=fbWriteCmd.bError;
nErrId :=fbWriteCmd.nErrId;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
4 EtherCAT Diagnostic

4.1 FB_EcGetAllSlaveAbnormalStateChanges

The function block FB_EcGetAllSlaveAbnormalStateChanges can be used to read the unexpected EtherCAT state changes of all the slaves connected to the master. If the call is successful, the buffer transferred in the parameter pBufAddr contains the number of unexpected state changes of all slaves as an array of UDINTs. EtherCAT state changes are unexpected if they were not requested by the EtherCAT master, e.g. if an EtherCAT slave spontaneously switches from OP state to SAFEOP state.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetId</td>
<td>String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)</td>
</tr>
<tr>
<td>pAddrBuf</td>
<td>Address of an array of UDINTs, into which the number of unexpected state changes of the individual slaves is to be written.</td>
</tr>
<tr>
<td>cbBufLen</td>
<td>Maximum available buffer size (in bytes) for the data to be read.</td>
</tr>
<tr>
<td>bExecute</td>
<td>The function block is activated by a positive edge at this input.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>Maximum time allowed for the execution of the function block.</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>This output is set when the function block is activated, and remains set until an acknowledgement is received.</td>
</tr>
<tr>
<td>bError</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>nErrId</td>
<td>Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.</td>
</tr>
<tr>
<td>nSlaves</td>
<td>The number of slaves connected to the master.</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 v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

TE1000

Version: 1.5

17
The FB_EcGetAllSlaveAddr function block allows the addresses of all the slaves connected to the master to be read. When the call is successful, the buffer passed in the parameter pAddrBuf contains the addresses of all the slaves as an array of UINTs.

VAR_INPUT

VAR_INPUT
sNetId : T_AmsNetId;
pAddrBuf : POINTER TO ARRAY[0..EC_MAX_SLAVES] OF UINT;
cbBufLen : UDINT
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
pAddrBuf: Address of an array of UINTs into which the addresses of the individual slaves are to be written.
cbBufLen: Maximum available buffer size (in bytes) for the data to be read.
bExecute: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
nSlaves : UINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.
bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.
nSlaves: The number of slaves connected to the master.

Example of an implementation in ST:

PROGRAM TEST_GetAllSlaveAddresses
VAR
 fbGetAllSlaveAddr : FB_EcGetAllSlaveAddr;
sNetId : T_AmsNetId := '172.16.2.131.2.1';
bExecute : BOOL;
slaveAddresses : ARRAY[0..255] OF UINT;
nSlaves : UINT := 0;
bError : BOOL;
nErrId : UDINT;
END_VAR

fbGetAllSlaveAddr(sNetId:= sNetId, pAddrBuf := ADR(slaveAddresses), cbBufLen:= SIZEOF(slaveAddresses), bExecute:=bExecute);
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2EtherCAT</td>
</tr>
</tbody>
</table>

4.3 **FB_EcGetAllCrcErrors**

The **FB_EcGetAllCrcErrors** function block allows the CRC error counters of all the slaves connected to the master to be read. The CRC errors at the individual ports of a slave are added.

In order to read the CRC errors of the individual ports (A, B and C) of a slave, it is necessary to call the **FB_EcGetSlaveCrcError** function block.

In order to read the CRC errors of the individual ports (A, B, C and D) of a slave, it is necessary to call the **FB_EcGetSlaveCrcErrorEx** function block.

**VAR_INPUT**

```plaintext
VAR_INPUT
  sNetId : T_AmsNetId;
  pCrcErrorBuf : POINTER TO ARRAY[0..EC_MAX_SLAVES] OF DWORD;
  cbBufLen : UDINT;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **pCrcErrorBuf**: The address of an array of DWORDs into which the CRC error counter is to be written.
- **cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  nSlaves : UINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.
- **nSlaves**: The number of slaves connected to the master.
**Example of an implementation in ST:**

PROGRAM TEST_GetAllSlaveCrcErrors
VAR
  fbGetAllSlaveCrcErrors : FB_EcGetAllSlaveCrcErrors;
  sNetId : T_AmsNetId := '172.16.2.131.2.1';
  bExecute : BOOL;
  crcErrors : ARRAY[0..255] OF DWORD;
  nSlaves : UINT := 0;
  bError : BOOL;
  nErrId : UDINT;
END_VAR

fbGetAllSlaveCrcErrors(sNetId:= sNetId, pCrcErrorBuf := ADR(crcErrors), cbBufLen:= SIZEOF(crcErrors) , bExecute:=bExecute);

nSlaves := fbGetAllSlaveCrcErrors.nSlaves;
bError := fbGetAllSlaveCrcErrors.bError;
nErrId := fbGetAllSlaveCrcErrors.nErrId;

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**4.4 **

**FB_EcGetAllSlavePresentStateChanges**

The function block `FB_EcGetAllSlavePresentStateChanges` can be used to read the EtherCAT state changes from state “slave is present” to “INIT_NO_COMM” of all slaves connected to the master. If the call is successful, the buffer transferred in the parameter `pAddrBuf` contains the number of state changes of all slaves as an array of UDINTs. The EtherCAT state change from state “slave is present” to “INIT_NO_COMM” means that the connection to the slave has been interrupted. For example by disconnecting the EtherCAT cable.

**VAR_INPUT**

VAR_INPUT
  sNetId : T_AmsNetId; (*AmsNetId of the EtherCAT master device*)
  pAddrBuf : POINTER TO ARRAY [0..EC_MAX_SLAVES] OF UDINT; (*Contains the address of the buffer the counters for the state changes from Present to INIT_NO_COMM to Present are copied to.*)
  cbBufLen : UDINT; (*Size of the buffer pAddrBuf. The size of the buffer must be at least nSlave *4 Bytes.*)
  bExecute : BOOL; (*Function Block execution is triggered by a rising edge at this input*)
  tTimeout : TIME; (*States the time before the function is cancelled.*)
END_VAR

**VAR_OUTPUT**

VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
pAddrBuf: Address of an array of UDINTs, into which the number of state changes from “slave is present” to INIT_NO_COMM for the individual slaves is to be written.

cbBufLen: Maximum available buffer size (in bytes) for the data to be read.
bExecute: The function block is activated by a positive edge at this input.
tTimeout: Maximum time allowed for the execution of the function block.
bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.

nSlaves: The number of slaves connected to the master.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.5 FB_EcGetConfSlaves

The function block FB_EcGetConfSlaves can be used to read a list of configured slaves from the EtherCAT master object directory.

VAR_INPUT

VAR_INPUT
sNetId : T_AmsNetId;
pArrEcConfSlaveInfo : POINTER TO ARRAY[0..EC_MAX_SLAVES] OF ST_EcSlaveConfigData;
cbBufLen : UDINT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

pArrEcConfSlaveInfo: Address of an array of structures of type ST_EcSlaveConfigData [102], into which data of each configured slave are to be written.

cbBufLen: Maximum available buffer size (in bytes) for the data to be read.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
nSlaves : UINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.

nSlaves: Returns the number of configured slaves.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.6 FB_EcGetLastProtErrInfo

The function block FB_EcGetLastProtErrInfo can be used to read additional error information relating to the most recent mailbox protocol error. An error-free mailbox command resets the last error every time.

VAR_INPUT

VAR_INPUT

| sNetId : T_AmsNetId; |
| nSlaveAddr : UINT; |
| eProtocol : E_EcMbxProtType := eEcMbxProt_FoE; |
| bExecute : BOOL; |
| tTimeout : TIME := DEFAULT_ADS_TIMEOUT; |
| sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId) |
| nSlaveAddr: Fixed address of the EtherCAT slave, whose error information is to be read. |
| eProtocol: EtherCAT mailbox protocol type [»100]. |
| bExecute: The function block is activated by a positive edge at this input. |
| tTimeout: Maximum time allowed for the execution of the function block. |

VAR_OUTPUT

VAR_OUTPUT

| bBusy : BOOL; |
| bError : BOOL; |
| nErrId : UDINT; |
| info : ST_EcLastProtErrInfo; |
| bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received. |
| bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command. |
| nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set. |
| info: Structure with additional error information [»101]. |
Sample in ST:

A rising edge at bGet triggers reading of additional error information relating to the most recent mailbox protocol error.

```plaintext
PROGRAM MAIN
VAR
  fbGetInfo : FB_EcGetLastProtErrInfo := ( sNetID := '172.16.6.195.2.1',
    nSlaveAddr := 1004,
    eProtocol := eEcMbxProt_FoE,
    tTimeout := DEFAULT_ADS_TIMEOUT );
  bGet : BOOL;
  bBusy : BOOL;
  bError : BOOL;
  nErrID : UDINT;
  sInfo : T_MaxString;
END_VAR

fbGetInfo( bExecute:= bGet,
            bBusy=>bBusy,
            bError=>bError,
            nErrId=>nErrId );

sInfo := BYTEARR_TO_MAXSTRING( fbGetInfo.info.binDesc );
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.7 FB_EcGetMasterDevState

The function block FB_EcGetMasterDevState can be used to read the current state of the EtherCAT master.

VAR_INPUT

```plaintext
VAR_INPUT
  sNetId   : T_AmsNetId;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

```plaintext
VAR_OUTPUT
  bBusy    : BOOL;
  bError   : BOOL;
  nErrId   : UDINT;
  nDevState : WORD;
END_VAR
```

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

nDevState: Current state of the master device.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.8 FB_EcGetScannedSlaves

The function block FB_EcGetScannedSlaves can be used to read a list of the currently available (scanned) slaves from the EtherCAT master object directory. To this end an online scan is executed, during which the EEPROMs of the EtherCAT slaves are read. The scanning process may take some time, depending on the number of connected slaves.

VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute  : BOOL;</td>
</tr>
<tr>
<td>sNetId    : T_AmsNetId;</td>
</tr>
<tr>
<td>pArrEcScannedSlaveInfo : POINTER TO ARRAY[0..EC_MAX_SLAVES] OF ST_EcSlaveScannedData;</td>
</tr>
<tr>
<td>cbBufLen  : UDINT;</td>
</tr>
<tr>
<td>tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

bExecute: The function block is activated by a positive edge at this input.

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

pArrEcScannedSlaveInfo: Address of an array of structures of type ST_EcSlaveScannedData [103], to which the data for each scanned slave are to be written.

cbBufLen: Maximum available buffer size (in bytes) for the data to be read.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy      : BOOL;</td>
</tr>
<tr>
<td>bError     : BOOL;</td>
</tr>
<tr>
<td>nErrId     : UDINT;</td>
</tr>
<tr>
<td>nSlaves    : UINT;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.

nSlaves: Returns the number of scanned slaves.
4.9 FB_EcGetSlaveCount

The function block FB_EcGetSlaveCount can be used to determine the number of slaves that are connected to the master.

**VAR_INPUT**

```plaintext
VAR_INPUT
  sNetId : T_AmsNetId;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  nSlaves : UINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
- **nSlaves**: The number of slaves connected to the master.

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_GetSlaveCount
VAR
  fbGetSlaveCount : FB_EcGetSlaveCount;
  sNetId : T_AmsNetId := '172.16.2.131.2.1';
  bExecute : BOOL;
  nSlaves : UINT;
  bError : BOOL;
  nErrId : UDINT;
END_VAR

fbGetSlaveCount(sNetId:= sNetId, bExecute:=bExecute);
  nSlaves := fbGetSlaveCount.nSlaves;
  bError := fbGetSlaveCount.bError;
  nErrId := fbGetSlaveCount.nErrId;
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</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 v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 4.10 FB_EcGetSlaveCrcError

The function block FB_EcGetSlaveCrcError allows the CRC error counters of the individual ports (A, B and C) of a slave to be read. If the call is successful, the output variable crcError, whose type is ST_EcCrcError, contains the requested CRC error counter.

The function block FB_EcGetSlaveCrcError can only be used with slaves with up to 3 ports (e.g. EK1100). The function block FB_EcGetSlaveCrcErrorEx can also be used with slaves with up to 4 ports (e.g. EK1122).

**VAR_INPUT**

```plaintext
sNetId : T_AmsNetId;
nSlaveAddr : UINT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **nSlaveAddr**: Fixed address of the EtherCAT slave whose CRC error counter is to be read
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
crcError : ST_EcCrcError;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
- **crcError**: CRC error counters for the individual ports.

**Example of an implementation in ST**:

```plaintext
PROGRAM TEST_GetSlaveCrcError
VAR
  fbGetSlaveCrcError : FB_EcGetSlaveCrcError;
sNetId : T_AmsNetId := '172.16.2.131.2.1';
bExecute : BOOL;
crcError : ST_EcCrcError;
```
EtherCAT Diagnostic

nSlaveAddr : UINT := 1001;
bError : BOOL;
nErrId : UDINT;
END_VAR

fbGetSlaveCrcError(sNetId:= sNetId, nSlaveAddr:= nSlaveAddr, bExecute:=bExecute);
crcError := fbGetSlaveCrcError.crcError;
bError := fbGetSlaveCrcError.bError;
nErrId := fbGetSlaveCrcError.nErrId;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.11 **FB_EcGetSlaveCrcErrorEx**

The function block **FB_EcGetSlaveCrcErrorEx** allows the CRC error counters of the individual ports (A, D, B and C) of a slave to be read. If the call is successful, the output variable crcError, whose type is **ST_EcCrcErrorEx**, contains the requested CRC error counter.

The function block **FB_EcGetSlaveCrcErrorEx** can also be used with slaves with up to 4 ports (e.g. EK1122). The function block **FB_EcGetSlaveCrcError** can only be used with slaves with up to 3 ports (e.g. EK1100).

VAR_INPUT

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId; (<em>AmsNetId of the EtherCAT master device</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlaveAddr</td>
<td>UINT; (<em>Address of the slave device</em>)</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL; (<em>Function block execution is triggered by a rising edge at this input.</em>)</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME; (<em>States the time before the function is cancelled.</em>)</td>
</tr>
</tbody>
</table>

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nErrId</td>
<td>UDINT;</td>
</tr>
<tr>
<td>CrcError</td>
<td>ST_EcCrcErrorEx; (<em>Crc error of the EtherCAT slave device</em>)</td>
</tr>
</tbody>
</table>

**sNetId**: String containing the AMS network ID of the EtherCAT master device (type: T_AmsNetID).

**nSlaveAddr**: Fixed address of the EtherCAT slave whose CRC error counter is to be read.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**CrcError**: CRC error counters for the individual ports. (type: **ST_EcCrcErrorEx [101]**)

---

TE1000

Version: 1.5

27
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.12 **FB_EcGetSlaveIdentity**

The function block **FB_EcGetSlaveIdentity** can be used to read the CANopen identity of an individual EtherCAT slave device. If the call is successful, the output variable identity, whose type is `ST_EcSlaveIdentity`, contains the requested identity information.

### VAR_INPUT

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type T_AmsNetId)
- **nSlaveAddr**: Fixed address of the EtherCAT slave.
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

### VAR_OUTPUT

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
- **identity**: CANopen identity [103] of the EtherCAT device.

**Example of an implementation in ST:**

```st
PROGRAM TEST_GetSlaveIdentity
VAR
  fbGetSlaveIdentity : FB_EcGetSlaveIdentity;
  sNetId : T_AmsNetId := '172.16.2.131.2.1';
  bExecute : BOOL;
  identity : ST_EcSlaveIdentity;
  nSlaveAddr : UINT := 1001;
  bError : BOOL;
  nErrId : UDINT;
END_VAR

fbGetSlaveIdentity(sNetId:= sNetId, nSlaveAddr:= nSlaveAddr, bExecute:=bExecute);
```
identity := fbGetSlaveIdentity.identity;
bError := fbGetSlaveIdentity.bError;
nErrId := fbGetSlaveIdentity.nErrId;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.13 **FB_EcGetSlaveTopologyInfo**

The function block **FB_EcGetSlaveTopologyInfo** can be used to determine topology information.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId; (<em>AmsNetId of the EtherCAT master device</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAddrBuf</td>
<td>POINTER TO ARRAY [0..EC_MAX_SLAVES] OF ST_TopologyDataEx; (<em>Contains the address of the buffer the topology data are copied to.</em>)</td>
</tr>
<tr>
<td>cbBufLen</td>
<td>UDINT; (<em>Size of the buffer pAddrBuf. The size of the buffer must be at least nSlave * 64 Bytes</em>)</td>
</tr>
<tr>
<td>nSlave * 64 Bytes*</td>
<td></td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL; (<em>Function block execution is triggered by a rising edge at this input</em>)</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME; (<em>States the time before the function is cancelled</em>)</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

| bBusy                   | BOOL;                                                                                 |
| bError                  | BOOL;                                                                                 |
| nErrId                  | UDINT;                                                                                 |
| nSlaves                 | UINT;                                                                                 |

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**pAddrBuf**: Address of an array of structures of type **ST_TopologyDataEx**[107], which contains the topology data.

**cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
**4.14 FB_EcMasterFrameCount**

The function block **FB_EcMasterFrameCount** can be used to determine the number of EtherCAT frames configured in the master.

**VAR_INPUT**

VAR_INPUT
- sNetId : T_AmsNetId;
- bExecute : BOOL;
- tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

VAR_OUTPUT
- bBusy : BOOL;
- bError : BOOL;
- nErrId : UDINT;
- nFrames : UDINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

nFrames: Number of EtherCAT frames.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**4.15 FB_EcMasterFrameStatistic**

The function block **FB_EcMasterFrameStatistic** can be used to read the frame statistics of the EtherCAT master. A distinction is made between cyclic and acyclic (queued) frames. Acyclic frames are used for the initialization or for parameter access to EtherCAT slaves. Frames are regarded as lost if they fail
to return to the master or are invalid.
The number of lost frames (i.e. lost or invalid cyclic frames), the number of cyclic frames per second, the
number of lost queued frames (i.e. lost or invalid acyclic frames) and the number of queued frames per
second is provided at the function block output.

VAR_INPUT

VAR_INPUT
  sNetId : T_AmsNetId;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
bExecute: The function block is activated by a positive edge at this input.
tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  nLostFrames : UDINT;
  fFramesPerSecond : LREAL;
  nLostQueuedFrames : UDINT;
  fQueuedFramesPerSecond : LREAL;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is
received.
bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of
the command.
nErrId: Supplies the ADS error code associated with the most recently executed command if the bError
output is set.
nLostFrames: Returns the current number of lost or invalid cyclic frames.
fFramesPerSecond: Returns the current number of cyclic frames per second.
nLostQueuedFrames: Returns the current number of lost or invalid queued (acyclic) frames.
fQueuedFramesPerSecond: Returns the current number of queued (acyclic) frames per second.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.16   FB_EcMasterFrameStatisticClearCRC

The function block FB_EcMasterFrameStatisticClearCRC can be used to delete the CRC error
counters of all EtherCAT slaves.
EtherCAT Diagnostic

VAR_INPUT

VAR_INPUT
  sNetId: T_AmsNetId;
  bExecute: BOOL;
  tTimeout: TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  nErrId: UDINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

4.17 FB_EcMasterFrameStatisticClearFrames

The function block FB_EcMasterFrameStatisticClearFrames can be used to delete the lost frame counters.

VAR_INPUT

VAR_INPUT
  sNetId: T_AmsNetId;
  bExecute: BOOL;
  tTimeout: TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  nErrId: UDINT;
END_VAR
**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 4.18 FB_EcMasterFrameStatisticClearTxRxErr

The function block FB_EcMasterFrameStatisticClearTxRxErr can be used to delete the error counters of the miniport driver of the network card.

**VAR_INPUT**

```plaintext
VAR_INPUT

sNetId : T_AmsNetId;
nEcMasterDevID : INT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the CPU (PC). (type: T_AMSNetId)
- **nEcMasterDevID**: Device ID of the EtherCAT master.
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
The function `F_CheckVendorId` returns TRUE, if the VendorID is Beckhoff, otherwise FALSE.

**VAR_INPUT**

```
VAR_INPUT
  stSlaveIdentity : ST_EcSlaveIdentity;
END_VAR
```

`stSlaveIdentity`: Slave identity, which can be read with `FB_EcGetSlaveIdentity[
`]

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
5 EtherCAT State Machine

5.1 FB_EcGetAllSlaveStates

The FB_EcGetAllSlaveStates function block allows the EtherCAT status and the Link status of all the slaves connected to the master to be read. When the call is successful, the buffer passed in the parameter pStateBuf contains the requested status information as an array of ST_EcSlaveState.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pStateBuf</td>
<td>POINTER TO ARRAY[0..EC_MAX_SLAVES] OF ST_EcSlaveState;</td>
</tr>
<tr>
<td>cbBufLen</td>
<td>UDINT;</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME := DEFAULT_ADS_TIMEOUT;</td>
</tr>
</tbody>
</table>

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**pStateBuf**: The address of an array of ST_EcSlaveStates [104] into which the slave states are to be written.

**cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nErrId</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nSlaves</td>
<td>UINT;</td>
</tr>
</tbody>
</table>

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set. Error 1798 (0x706) indicates a null pointer at the buffer address. Error 1797 (0x705) indicates inadequate buffer size.

**nSlaves**: The number of slaves connected to the master.

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_GetAllSlaveStates
VAR
  fbGetAllSlaveStates : FB_EcGetAllSlaveStates;
  sNetId : T_AmsNetId := '172.16.2.131.2.1';
  bExecute : BOOL;
  devStates : ARRAY[0..255] OF ST_EcSlaveState;
  nSlaves : UINT := 0;
  bError : BOOL;
  nErrId : UDINT;
END_VAR
```
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

5.2 FB_EcGetMasterState

The function block FB_EcGetMasterState can be used to read the EtherCAT state of the master. If the call is successful, the State output variable of type WORD contains the requested status information.

VAR_INPUT

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute</td>
<td>BOOL</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
</tr>
</tbody>
</table>

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>nErrId</td>
<td>UDINT</td>
</tr>
<tr>
<td>state</td>
<td>WORD</td>
</tr>
</tbody>
</table>

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

state: Current EtherCAT state of the master. The possible values are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Master is in Init state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Master is in Pre-operational state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Master is Safe-operational state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_OP</td>
<td>0x08</td>
<td>Master is Operational state</td>
</tr>
</tbody>
</table>
Example of an implementation in ST:

```plaintext
PROGRAM TEST_GetMasterState
VAR
  fbGetMasterState : FB_EcGetMasterState;
  sNetId           : T_AmsNetId := '172.16.2.131.2.1';
  bExecute         : BOOL;
  state            : WORD;
  bError           : BOOL;
  nErrId           : UDINT;
END_VAR
fbGetMasterState(sNetId:= sNetId, bExecute:=bExecute);
state := fbGetMasterState.state;
bError := fbGetMasterState.bError;
nErrId := fbGetMasterState.nErrId;
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

5.3 FB_EcGetSlaveState

The function block FB_EcGetSlaveState allows the EtherCAT status and the Link status of an individual EtherCAT slave to be read. If the call is successful, the output variable state, whose type is ST_EcSlaveState, contains the requested status information.

VAR_INPUT

- **sNetId** : String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **nSlaveAddr** : Fixed address of the EtherCAT slave whose status is to be read
- **bExecute** : The function block is activated by a positive edge at this input.
- **tTimeout** : Maximum time allowed for the execution of the function block.

VAR_OUTPUT

- **bBusy** : This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError** : This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId** : Supplies the ADS error code associated with the most recently executed command if the bError output is set.
**EtherCAT State Machine**

**state**: Structure that contains the EtherCAT status and the Link status of the slave. (type: ST_EcSlaveState)

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_GetSlaveState
VAR
    fbGetSlaveState : FB_EcGetSlaveState;
    sNetId          : T_AmsNetId := '172.16.2.131.2.1';
    bExecute        : BOOL;
    state           : ST_EcSlaveState;
    nSlaveAddr      : UINT := 1001;
    bError          : BOOL;
    nErrId          : UDINT;
END_VAR

fbGetSlaveState(sNetId:= sNetId, nSlaveAddr:= nSlaveAddr, bExecute:=bExecute);
state := fbGetSlaveState.state;
bError := fbGetSlaveState.bError;
nErrId := fbGetSlaveState.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**5.4 FB_EcReqMasterState**

With this function block the EtherCAT state of a master device can be requested and set. The requested EtherCAT state is transferred in the state variable. The function block becomes inactive as soon as it has requested the EtherCAT state. Unlike the function block FB_EcSetMasterState it does not wait until the new state is set.

See also: FB_EcSetMasterState

**VAR_INPUT**

```plaintext
VAR_INPUT
    sNetId    : T_AmsNetId;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
    state    : WORD;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.
- **state**: EtherCAT state requested from the master. The possible State values are:
### EtherCAT State Machine

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Request Init state from master</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Request Pre-operational state from master</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Request Safe-operational state from master</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_OP</td>
<td>0x08</td>
<td>Request Operational state from master</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
    bBusy : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_ReqMasterState
VAR
    fbReqMasterState : FB_EcReqMasterState;
    sNetId := T_AmsNetId:= '172.16.2.131.2.1';
    bExecute : BOOL;
    state : WORD := EC_DEVICE_STATE_INIT;
    bError : BOOL;
    nErrId : UDINT;
END_VAR

    fbReqMasterState(sNetId:= sNetId, bExecute:=bExecute, state:=state);
    bError := fbGetMasterState.bError;
    nErrId := fbGetMasterState.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**5.5 FB_EcReqSlaveState**

With this function block a slave can be set to a specified EtherCAT state. The requested EtherCAT state is transferred in the state variable. The function block becomes inactive as soon as it has sent the command to change state. Unlike the function block FB_EcSetSlaveState it does not wait until the EtherCAT slave has attained the new state.

See also: FB_EcSetSlaveState [42]
VAR_INPUT
sNetId : T_AmsNetId;
nSlaveAddr : UINT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
state : WORD;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

nSlaveAddr: Fixed address of the EtherCAT slave whose EtherCAT state is to be set.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

state: EtherCAT state to which the slave is to be set. The possible State values are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Set slave to Init state</td>
</tr>
<tr>
<td>EC DEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Set slave to Pre-operational state</td>
</tr>
<tr>
<td>EC DEVICE_STATE_BOOTSTRAP</td>
<td>0x03</td>
<td>Set slave to Bootstrap state. This state is used for firmware downloads.</td>
</tr>
<tr>
<td>EC DEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Set slave to Safe-operational state</td>
</tr>
<tr>
<td>EC DEVICE_STATE_OP</td>
<td>0x08</td>
<td>Set slave to Operational state</td>
</tr>
<tr>
<td>EC DEVICE_STATE_ERROR</td>
<td>0x10</td>
<td>If the error bit in the status byte is set in the EtherCAT slave (state.deviceState &amp; EC_DEVICE_STATE_ERROR = TRUE), the error bit can be reset by setting EC_DEVICE_STATE_ERROR.</td>
</tr>
</tbody>
</table>

VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

Example of an implementation in ST:

PROGRAM TEST_RqSlaveState
VAR
  fbGetSlaveState : FB_EcReqSlaveState;
  sNetId : T_AmsNetId:= '172.16.2.131.2.1';
bExecute : BOOL;
state : WORD := EC DEVICE_STATE_INIT;
nSlaveAddr : UINT := 1001;
bError : BOOL;
nErrId : UDINT;
END_VAR

fbGetSlaveState(sNetId:= sNetId, nSlaveAddr:= nSlaveAddr, bExecute:=bExecute, state:=state);
bError := fbGetSlaveState.bError;
nErrId := fbGetSlaveState.nErrId;
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

5.6  FB_EcSetMasterState

With this function block the EtherCAT state of a master device can be requested and set. The requested EtherCAT state is transferred with the reqState variable. The function block requests the EtherCAT state and, unlike the function block FB_EcReqMasterState, remains active until the new state is set or the maximum time tTimeout is exceeded. The current state is output in the currState variable.

See also: FB_EcReqMasterState

VAR_INPUT

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME := T#10s;</td>
</tr>
<tr>
<td>reqState</td>
<td>WORD;</td>
</tr>
</tbody>
</table>

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

reqState: EtherCAT state requested from the master. The possible values for reqState are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Request Init state from master</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Request Pre-operational state from master</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Request Safe-operational state from master</td>
</tr>
<tr>
<td>EC DEVICE_STATE_OP</td>
<td>0x08</td>
<td>Request Operational state from master</td>
</tr>
</tbody>
</table>

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nErrId</td>
<td>UDINT;</td>
</tr>
<tr>
<td>currState</td>
<td>WORD;</td>
</tr>
</tbody>
</table>

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
**currState**: Current EtherCAT state of the master.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 5.7 FB_EcSetSlaveState

With this function block a slave can be set to a specified EtherCAT state. The requested EtherCAT state is transferred with the reqState variable. The function block sends the command to change state and, unlike the function block FB_EcRegSlaveState, remains active until the EtherCAT slave has attained the new state or the maximum time tTimeout is exceeded. The current state is output in the currState variable.

See also: [FB_EcReqSlaveState](#) [39]

**VAR_INPUT**

<table>
<thead>
<tr>
<th>sNetId</th>
<th>T_AmsNetId;</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlaveAddr</td>
<td>UINT;</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME := T#10s;</td>
</tr>
<tr>
<td>reqState</td>
<td>WORD;</td>
</tr>
</tbody>
</table>

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**nSlaveAddr**: Fixed address of the EtherCAT slave whose EtherCAT state is to be set.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**reqState**: EtherCAT state to which the slave is to be set. The possible values for reqState are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Set slave to Init state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Set slave to Pre-operational state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_BOOTSTRAP</td>
<td>0x03</td>
<td>Set slave to Bootstrap state. This state is used for firmware downloads.</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Set slave to Safe-operational state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_OP</td>
<td>0x08</td>
<td>Set slave to Operational state</td>
</tr>
<tr>
<td>EC_DEVICE_STATE_ERROR</td>
<td>0x10</td>
<td>If the error bit in the status byte is set in the EtherCAT slave (currState.deviceState AND EC_DEVICE_STATE_ERROR = TRUE), the error bit can be reset by setting EC_DEVICE_STATE_ERROR.</td>
</tr>
</tbody>
</table>
VAR_OUTPUT
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  currState : ST_EcSlaveState;
END_VAR

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**currState**: Current EtherCAT state [104] of the slave.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
6 CoE interface

6.1 FB_EcCoeSdoRead

The **FB_EcCoeSdoRead** function block allows data to be read from an object dictionary of an EtherCAT slave through an SDO (Service Data Object) access. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The nSubIndex and nIndex parameters allow the object that is to be read to be selected. The function block **FB_EcCoeSdoReadEx** must be used for access to the complete parameter, including subelements.

**VAR_INPUT**

```plaintext
VAR_INPUT
sNetId : T_AmsNetId;
nSlaveAddr : UINT;
nSubIndex : BYTE;
nIndex : WORD;
pDstBuf : PVOID;
cbBufLen : UDINT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **nSlaveAddr**: Fixed address of the EtherCAT slave to which the SDO upload command should be sent.
- **nSubIndex**: Subindex of the object that is to be read.
- **nIndex**: Index of the object that is to be read.
- **pDstBuf**: Address (pointer) to the receive buffer.
- **cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.
- **bExecute**: The function block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
**Example of an implementation in ST:**

```st
PROGRAM TEST_SdoRead
VAR
    fbSdoRead : FB_EcCoESdoRead;
    sNetId : T_AmsNetId := '172.16.2.131.2.1';
    bExecute : BOOL;
    nSlaveAddr : UINT := 1006;
    nIndex : WORD := 16#1018;
    nSubIndex : BYTE := 1;
    vendorId : UDINT;
    bError : BOOL;
    nErrId : UDINT;
END_VAR

fbSdoRead(sNetId:= sNetId, nSlaveAddr := nSlaveAddr, nIndex:= nIndex, nSubIndex := nSubIndex, pDstBuf:= ADR(vendorId), cbBufLen:= SIZEOF(vendorId), bExecute:= bExecute);

bError:= fbSdoRead.bError;

nErrId:= fbSdoRead.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 6.2 **FB_EcCoESdoReadEx**

The `FB_EcCoESdoReadEx` function block allows data to be read from an object dictionary of an EtherCAT slave through an SDO (Service Data Object) access. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The `nSubIndex` and `nIndex` parameters allow the object that is to be read to be selected. The parameter with subelements can be read via `bCompleteAccess := TRUE`.

**VAR_INPUT**

```st
VAR_INPUT
    sNetId : T_AmsNetId; (* AmsNetId of the EtherCAT master device.*)
    nSlaveAddr : UINT; (* Address of the slave device.*)
    nSubIndex : BYTE; (* CANopen Sdo subindex.*)
    nIndex : WORD; (* CANopen Sdo index.*)
    pDstBuf : PVOID; (* Contains the address of the buffer for the received data.*)
    cbBufLen : UDINT; (* Contains the max. number of bytes to be received.*)
    bExecute : BOOL; (* Function block execution is triggered by a rising edge at this input.*)
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled.*)
    bCompleteAccess : BOOL; (* access complete object*)
END_VAR
```

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**nSlaveAddr**: Fixed address of the EtherCAT slave to which the SDO upload command should be sent.

**nSubIndex**: Subindex of the object that is to be read.

**nIndex**: Index of the object that is to be read.

**pDstBuf**: Address (pointer) to the receive buffer.
cbBufLen: Maximum available buffer size (in bytes) for the data to be read.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

bCompleteAccess: If bCompleteAccess is set, the whole parameter can be read in a single access.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
</tr>
<tr>
<td>bError</td>
</tr>
<tr>
<td>nErrId</td>
</tr>
</tbody>
</table>

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetId</td>
</tr>
<tr>
<td>nSlaveAddr</td>
</tr>
<tr>
<td>nSubIndex</td>
</tr>
<tr>
<td>nIndex</td>
</tr>
<tr>
<td>pSrcBuf</td>
</tr>
<tr>
<td>cbBufLen</td>
</tr>
<tr>
<td>bExecute</td>
</tr>
<tr>
<td>tTimeout</td>
</tr>
</tbody>
</table>

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 6.3 FB_EcCoeSdoWrite

The **FB_EcCoeSdoWrite** function block permits an object from the object directory of an EtherCAT slave to be written by means of an SDO download. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The nSubIndex and nIndex parameters allow the object that is to be written to be selected. The function block **FB_EcCoeSdoWriteEx** must be used for access to the complete parameter, including subelements.

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**nSlaveAddr**: Fixed address of the EtherCAT slave to which the SDO download command should be sent.

**nSubIndex**: Subindex of the object that is supposed to be written.
**nIndex**: Index of the object that is supposed to be written.

**pSrcBuf**: Address (pointer) to the send buffer.

**cbBufLen**: Number of date to be sent in bytes.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**VAR OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
<th></th>
<th></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>nErrId</td>
<td>UDINT</td>
<td></td>
</tr>
</tbody>
</table>

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_SdoWrite

VAR
  fbSdoWrite : FB_EcCoESdoWrite;
  sNetId      := '172.16.2.131.2.1'; (* NetId of EtherCAT Master *)
  nSlaveAddr  := 1005; (* Port Number of EtherCAT Slave *)
  nIndex      := 16#4062; (* CoE Object Index *)
  nSubIndex   := 1; (* Subindex of CoE Object *)
  nValue      := 2; (* variable to be written to the CoE Object *)
  bExecute    := TRUE;
  bError      := FALSE;
  nErrId      := 0;
END_VAR

fbSdoWrite(
  sNetId     := sNetId,
  nSlaveAddr := nSlaveAddr,
  nIndex     := nIndex,
  nSubIndex  := nSubIndex,
  pSrcBuf    := ADR(nValue),
  cbBufLen   := SIZEOF(nValue),
  bExecute   := bExecute);

IF NOT fbSdoWrite.bBusy THEN
  bExecute := FALSE;
IF NOT fbSdoWrite.bError THEN
  (* write successful *)
  bError := FALSE;
  nErrId := 0;
ELSE
  (* write failed *)
  bError := fbSdoWrite.bError;
  nErrId := fbSdoWrite.nErrId;
END_IF
fbSdoWrite(bExecute := FALSE);
END_IF
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
The **FB_EcCoeSdoWriteEx** function block permits an object from the object directory of an EtherCAT slave to be written by means of an SDO download. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The nSubIndex and nIndex parameters allow the object that is to be written to be selected. Via `bCompleteAccess := TRUE` the parameter can be written with sub-elements.

### VAR_INPUT

- **sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
- **nSlaveAddr**: Fixed address of the EtherCAT slave to which the SDO download command should be sent.
- **nSubIndex**: Subindex of the object that is supposed to be written.
- **nIndex**: Index of the object that is supposed to be written.
- **pSrcBuf**: Address (pointer) to the send buffer.
- **cbBufLen**: Number of date to be sent in bytes.
- **tTimeout**: Maximum time allowed for the execution of the function block.
- **bCompleteAccess**: If `bCompleteAccess` is set, the whole parameter can be written in a single access.

### VAR_OUTPUT

- **bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.
- **bError**: This output is set after the `bBusy` output has been reset when an error occurs in the transmission of the command.
- **nErrId**: Supplies the ADS error code associated with the most recently executed command if the `bError` output is set.
6.5 **FB_CoERead_ByDriveRef**

The function block **FB_CoERead_ByDriveRef** can be used to read drive parameters by means of the “CANopen over EtherCAT (CoE)” protocol. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The nSubIndex and nIndex parameters allow the object that is to be read to be selected. Via `bCompleteAccess := TRUE` the parameter can be read with sub-elements.

**VAR_INPUT**

- `stDriveRef` : `ST_DriveRef`; (*Contains sNetID of EcMaster, nSlaveAddr of EcDrive, nDriveNo of EcDrive, either preset or read from NC*)
- `nIndex` : `WORD`; (*SoE IDN: e.g. “S_0_IDN+1” for S-0-0001 or “P_0_IDN+23” for P-0-0023*)
- `nSubIndex` : `BYTE`;  (*Contains the address of the buffer for the received data*)
- `pDestBuf` : `POVOID`;  (*Contains the address of the buffer for the received data*)
- `cbBufLen` : `UDINT`; (*Contains the max. number of bytes to be received*)
- `bExecute` : `BOOL`; (*Function block execution is triggered by a rising edge at this input*)
- `tTimeout` : `TIME`; (*States the time before the function is cancelled*)
- `bCompleteAccess` : `BOOL`;

**VAR_OUTPUT**

- `bBusy` : `BOOL`;
- `bError` : `BOOL`;
- `iAdsErrId` : `UINT`;
- `iCANopenErrId` : `UINT`;

**stDriveRef**: Structure containing the AMS network ID of the EtherCAT master device and the address of the slave device. The reference to the drive can be linked directly to the PLC in the System Manager. To this end an instance of `ST_PlcDriveRef` must be used and the NETID of the Byte array converted to a string.

**nIndex**: Index of the object that is to be read.

**nSubIndex**: Subindex of the object that is to be read.

**pDestBuf**: Address (pointer) to the receive buffer.

**cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**bCompleteAccess**: If `bCompleteAccess` is set, the whole parameter can be read in a single access.

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the `bBusy` output has been reset when an error occurs in the transmission of the command.

**iAdsErrId**: In the case of a set `bError` output returns the ADS error code of the last executed command.
CoE interface

iCANopenErrId: Returns the CANopen error code if the bError output is set.

Example of an implementation in ST:

```st
PROGRAM MAIN
VAR
  fbCoERead : FB_CoERead_ByDriveRef;
  stDriveRef : ST_DriveRef;
  nIndex : WORD := 16#1018;
  nSubIndex : BYTE := 1;
  bExecute : BOOL := TRUE;
  tTimeout : TIME := T#5S;
  bCompleteAccess : BOOL := TRUE;
  vendorId : UDINT;
  bError : BOOL;
  nAdsErrId : UDINT;
  nCANopenErrId : UDINT;
END_VAR

fbCoERead(
  stDriveRef:= stDriveRef,
  nIndex:= nIndex,
  nSubIndex:= nSubIndex,
  pDstBuf:= ADR(vendorId),
  cbBufLen:= SIZEOF(vendorId),
  bExecute:= bExecute,
  tTimeout:= tTimeout,
  bCompleteAccess:= bCompleteAccess,
);
IF NOT fbCoERead.bBusy THEN
  bError:=fbCoERead.bError;
  nAdsErrId:=fbCoERead.iAdsErrId;
  nCANopenErrId:=fbCoERead.iCANopenErrId;
  bExecute := FALSE;
  fbCoERead(bExecute := bExecute);
END_IF
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

6.6 FB_CoEWrite_ByDriveRef

The function block FB_CoEWrite_ByDriveRef can be used to write drive parameters based on the “CANopen over EtherCAT (CoE)” protocol. This requires the slave to have a mailbox and to support the “CANopen over EtherCAT” (CoE) protocol. The nSubIndex and nIndex parameters allow the object that is to be written to be selected. Via bCompleteAccess := TRUE the parameter can be written with sub-elements.

VAR_INPUT
  stDriveRef : ST_DriveRef; (*Contains sNetID of EcMaster, nSlaveAddr EcDrive, nDriveNo of EcDrive, either preset or read from NC*)
  nIndex : WORD; (*SoE IDN: e.g. “S_0_IDN+1” for S-0-0001 or “P_0_IDN+23” for P-0-0023*)
  nSubIndex : BYTE; (*SoE element*)
  pSrcBuf : PVOID; (*Contains the address of the buffer containing the data to be sent*)
  cbBufLen : UDINT; (*Contains the max. number of bytes to be received*)
bExecute : BOOL; (*Function block execution is triggered by a rising edge at this input*)
tTimeout : TIME; (*States the time before the function is cancelled*)
bCompleteAccess : BOOL;
END_VAR

stDriveRef: Structure containing the AMS network ID of the EtherCAT master device and the address of the slave device. The reference to the drive can be linked directly to the PLC in the System Manager. To this end an instance of ST_PlcDriveRef must be used and the NETID of the Byte array converted to a string.

nIndex: Index of the object that is supposed to be written.
nSubIndex: Subindex of the object that is supposed to be written.
pSrcBuf: Address (pointer) to the send buffer.

cbBUFLen: Maximum available buffer size for the data to be sent in bytes.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

bCompleteAccess: If bCompleteAccess is set, the whole parameter can be read in a single access.

VAR_OUTPUT
    bBusy : BOOL;
    bError : BOOL;
    iAdsErrId : UINT;
    iCANopenErrId : UINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

iAdsErrId: In the case of a set bError output returns the ADS error code of the last executed command.

iCANopenErrId: Returns the CANopen error code if the bError output is set.

Example of an implementation in ST:

PROGRAM MAIN
VAR
    fbCoEWrite : FB_CoEWrite_ByDriveRef;
    stDriveRef : ST_DriveRef;
    nIndex : WORD := 16#1018;
    nSubIndex : BYTE := 1;
    bExecute : BOOL := TRUE;
    tTimeout : TIME := T#5S;
    bCompleteAccess : BOOL := TRUE;
    vendorId : UDINT := 2;
    bError : BOOL;
    nAdsErrId : UDINT;
    nCANopenErrId : UDINT;
END_VAR

    fbCoEWrite(
        stDriveRef := stDriveRef,
        nIndex := nIndex,
        nSubIndex := nSubIndex,
        pSrcBuf := ADR(vendorId),
        cbBUFLen := SIZEOF(vendorId),
        bExecute := bExecute,
        tTimeout := tTimeout,
        bCompleteAccess := bCompleteAccess,
    );

    IF NOT fbCoEWrite.bBusy THEN
        bError := fbCoEWrite.bError;
        nAdsErrId := fbCoEWrite.iAdsErrId;
        nCANopenErrId := fbCoEWrite.iCANopenErrId;
        bExecute := FALSE;
        fbCoEWrite(bExecute := bExecute);
    END_IF
## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
7 FoE interface

7.1 FB_EcFoeAccess

This function block writes or reads data via the communication port of the “File access over EtherCAT” mailbox protocol.

**VAR_INPUT**

hFoe : T_HFoe;
pBuffer : DWORD;
cbBuffer : UDINT;
bExecute : BOOL;
tTimeout : TIME := DEFAULT_ADS_TIMEOUT;

**VAR_OUTPUT**

bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
cbDone : UDINT;
bEOF : BOOL;

hFoe: “File access over EtherCAT” handle [p 110].

pBuffer: Contains the address of the buffer into which the data are to be read (read access) or the address of buffer containing the data to be written (write access). The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

cbBuffer: Contains the number of data bytes to be written or read.

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

cbDone: Number of the most recent successfully written or read data bytes.

bEOF: End of File. This variable becomes TRUE if the end of the file is reached during read access. For write access this variable has no purpose.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
7.2 FB_EcFoeClose

This function block closes the communication port for the “File access over EtherCAT” mailbox protocol.

VAR_INPUT

VAR_INPUT
  hFoe : T_HFoe;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

hFoe: “File access over EtherCAT” handle [110].

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

7.3 FB_EcFoeLoad

The function block FB_EcFoeLoad can be used to download or upload files to or from an EtherCAT device via the “File access over EtherCAT” mailbox protocol.
The path can only point to the local file system on the computer. This means that network paths cannot be used here! To upload or download files via the FoE protocol, the function block automatically resets the EtherCAT device to BOOTSTRAP mode. Finally, the function block tries to reset the device to the original state.

### VAR_INPUT

```plaintext
sNetId : T_AmsNetId;
nSlaveAddr : UINT;
SPathName : T_MaxString;
dwPass : DWORD := 0;
eMode : E_EcFoeMode := eFoeMode_Write;
bExecute : BOOL;
tTimeout : TIME := T#200s;

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
nSlaveAddr: Fixed address of the EtherCAT slave whose file is to be uploaded or downloaded.
SPathName: Contains the path and file names of the file to be written or read (e.g.: 'C:\FOE_Test\EL6751\ECATFW__EL6751_C6_V0030.efw').
dwPass: Password (default: 0).
eMode: “File access over EtherCAT” access mode [100] (default: write access).
bExecute: The function block is activated by a positive edge at this input.
tTimeout: Maximum time that must not be exceeded when the function block is executed (default: 200 s).
```

### VAR_OUTPUT

```plaintext
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
cbLoad : UDINT;
nProgress : UDINT;
sInfo : T_MaxString;

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.
bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.
nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.
cbLoad: Number of successfully written or read data bytes.
nProgress: Write access progress (range: 0 - 100%). This variable is currently not used for read access, in which case it is always 0.
sInfo: Additional command information as string (reserved).
```

### Sample in ST:

A rising edge at the bLoad variable triggers the firmware download via the “File access over EtherCAT” mailbox protocol.

```plaintext
PROGRAM MAIN
VAR
  fbDownload : FB_EcFoeLoad := {
    sNetID := '5.0.34.3.1',
    nSlaveAddr := 3004,
    SPathName := 'C:\FOE_Test\EL6751\ECATFW__EL6751_C6_V0030.efw',
    dwPass := 0,
    eMode := eFoeMode_Write };
```
VAR_INPUT

VAR_INPUT
  sNetId : T_AmsNetId;
  nPort : UINT;
  sPathName : T_MaxString;
  dwPass : DWORD;
  eMode : E_EcFoeMode;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

nPort: Fixed address of the EtherCAT device.

sPathName: Path name (e.g.: 'c:\TwinCAT\FOE\Data.fwp'). By default, only the file name (without file name extension) is extracted from the file path that was entered and used as the file name for the FoE protocol (in our example: 'Data'). From library version 3.3.12.0, file names including the file name extension can also be used (in our example: 'Data.fwp').

The use of the file name extension for all instances of the FB_EcFoeOpen function block can be enabled or disabled via the global Boolean variable Tc2_EtherCAT.bEcFoeOpenFileNameWithFileExt. By default, the variable has the value FALSE (no file name extension). If you set the value to TRUE, the use of the file name extensions is enabled.

Note that the FoE function blocks were originally used for firmware updates for which no file name extension was used. If you want to update the firmware, you may have to make sure that the global variable has its original default value, i.e. FALSE.

dwPass: Password.
eMode: Access mode [100] (write/read access).

bExecute: The function block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  hFoe : T_HFoe;
END_VAR

bBusy: This output is set when the function block is activated, and remains set until an acknowledgement is received.

bError: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

nErrId: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

hFoe: “File access over EtherCAT” handle [110].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

7.5 FB_EcFoeReadFile

The function block FB_EcFoeReadFile can be used to download files from an EtherCAT device to the local data carrier via the "File access over EtherCAT" mailbox protocol.

The path can only point to the local file system on the computer. This means that network paths cannot be used here!

VAR_INPUT

VAR_INPUT
  sFSrvNetId : T_AmsNetId DWORD := '';
  sFSrvPathName : T_MaxString;
  sEcNetId : T_AmsNetId;
  nSlaveAddr : UINT;
  sFoEPathName : T_MaxString;
  dwPass : DWORD := 0;
  bExecute : BOOL;
  tTimeout : TIME := T#200s;
END_VAR

sFSrvNetID: AMS network ID of the computer on which the file that was read is to be written (local computer). (type: T_AmsNetId)
**sFSrvPathName**: Contains the path and file name of the file to be written (e.g. 'C:\Data\LogData.csv'), (type: T_MaxString).

**sEcNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**nSlaveAddr**: Address of the EtherCAT slave (type: UINT)

**sFoEPathName**: Name of the file on the EtherCAT slave, e.g. 'LogData' (type: T_MaxString).

**dwPass**: Password (type: DWORD)

**bExecute**: The function block is activated by a positive edge at this input (type: BOOL).

**tTimeout**: Maximum time that must not be exceeded when the function block is executed (default: 200 s) (type: TIME).

### VAR_OUTPUT

```
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  cbRead : UDINT;
  sInfo : T_MaxString;
END_VAR
```

**bBusy**: This output is set when the function block is activated and remains set until an acknowledgement is received (type: BOOL).

**bError**: This output is set after the bBusy output has been reset if an error occurs during the transfer of the command (type: BOOL).

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set. (type: UDINT)

**cbRead**: Number of successfully read data bytes. (type: UDINT)

**sInfo**: Additional FoE error information (reserved) (type: T_MaxString).

### Sample in ST:

A rising edge at the bExecute variable triggers reading of the specified file via the "File access over EtherCAT" mailbox protocol. The file named in sFoEPathName is read by the selected EtherCAT slave (sEcNetId + nSlaveAddr). The file is stored on the selected computer (sFSrvNetID) under the name specified in sFSrvPathName. If a password is required for reading the file from the EtherCAT slave, this can be specified via dwPass.

The read and write operation is not completed until bBusy switches to FALSE. Only then are the error information or the number of bytes read to be evaluated.

```st
PROGRAM MAIN
VAR
  fbEcReadFile : FB_EcFoeReadFile := (
    sFSrvNetID := '5.0.34.1.1', (* NetID for target file *)
    sFSrvPathName := 'C:\Data\LogData.csv', (* Pathname for target file *)
    sEcNetId := '5.0.34.38.3.1', (* NetID of EtherCAT master *)
    nSlaveAddr := 1004, (* EtherCAT slave address *)
    sFoEPathName := 'LogData', (* Name of source file *)
    dwPass := 0
  );
  bExecute : BOOL := TRUE;
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
  nBytesRead : UDINT;
END_VAR

fbEcReadFile {
  bExecute := bExecute,
  bBusy => bBusy,
  bError => bError,
  nErrId => nErrId,
  cbRead => nBytesRead
```

---

**Version**: 1.5

**TE1000**
IF NOT bBusy THEN
  bExecute := FALSE;

IF NOT bError THEN
  (* done, no error *)
  nBytesRead := fbEcReadFile.cbRead;
ELSE
  (* evaluate error *)
  nBytesRead := 0;
END_IF

fbEcReadFile (bExecute := FALSE);
END_IF

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT &gt;= 3.3.14</td>
</tr>
</tbody>
</table>

Also see about this

E_EcFoeMode [100]
8 SoE interface

8.1 FB_EcSoeRead

The function block FB_EcSoeRead can be used to read drive parameters by means of the “Servo drive profile over EtherCAT (SoE)” protocol. To this end the slave must have a mailbox and support the SoE protocol. The drive parameter to be read is specified with the parameters nIdn (identification number), nElement and nDriveNo.

VAR_INPUT

VAR_INPUT
sNetId : T_AmsNetId;
SlaveAddr : UINT;
SubIndex : BYTE;
Idn : WORD;
Element : BYTE;
DriveNo : BYTE;
Command : BOOL
DstBuf : PVOID;
cbBufLen : UDINT;
Execute : BOOL;
Timeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR

sNetId: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)
nSlaveAddr: Fixed address of the EtherCAT slave to which the SoE read command is to be sent.
nIdn: Identification number of the parameter to be read.
nElement: Element number of the parameter to be read. The following values are permitted:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Data status</td>
</tr>
<tr>
<td>0x02</td>
<td>Name (read only)</td>
</tr>
<tr>
<td>0x04</td>
<td>Attribute</td>
</tr>
<tr>
<td>0x08</td>
<td>Unit</td>
</tr>
<tr>
<td>0x10</td>
<td>Minimum</td>
</tr>
<tr>
<td>0x20</td>
<td>Maximum</td>
</tr>
<tr>
<td>0x40</td>
<td>Value</td>
</tr>
<tr>
<td>0x80</td>
<td>Default</td>
</tr>
</tbody>
</table>

nDriveNo: Drive number.

bCommand: This parameter should be set if internal command execution is to be used.

pDstBuf: Address (pointer) to the receive buffer.

cbBufLen: Maximum available buffer size (in bytes) for the data to be read.

bExecute: The function block is activated by a positive edge at this input.
**tTimeout:** Maximum time allowed for the execution of the function block.

**VAR OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrId : UDINT;</td>
</tr>
</tbody>
</table>

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_SoERead
VAR
    fbSoERead : FB_EcSoERead;
    sNetId : T_AmsNetId := '172.16.2.131.2.1';
    bExecute : BOOL;
    nSlaveAddr : UINT := 1006;
    nIdn : WORD := 15;
    nElement : BYTE := 0;
    nDriveNo : BYTE := 0;
    bCommand : BOOL := FALSE;
    val : UINT;
    bError : BOOL;
    nErrId : UDINT;
END_VAR

fbSoERead(sNetId:= sNetId, nSlaveAddr := nSlaveAddr, nIdn := nIdn, nElement:= nElement, nDriveNo := nDriveNo, bCommand:= bCommand, pDstBuf:= ADR(val), cbBufLen:= SIZEOF(val), bExecute:= bExecute);

bError := fbSoERead.bError;
nErrId := fbSoERead.nErrId;
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**8.2 **FB_EcSoeWrite

The function block **FB_EcSoeWrite** can be used to write drive parameters by means of the “Servo drive profile over EtherCAT (SoE)” protocol. To this end the slave must have a mailbox and support the SoE protocol. The drive parameter to be written is specified with the parameters nIdn (identification number), nElement and nDriveNo.
**VAR_INPUT**

```plaintext
VAR_INPUT
  sNetId : T_AmsNetId;
  nSlaveAddr : UINT;
  nIdn : WORD;
  nElement : BYTE;
  nDriveNo : BYTE;
  pCommand : BOOL;
  pSrcBuf : PVOID;
  cbBufLen : UDINT;
  bExecute : BOOL;
  tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

**sNetId**: String containing the AMS network ID of the EtherCAT master device. (type: T_AmsNetId)

**nSlaveAddr**: Fixed address of the EtherCAT slave to which the SoE write command is to be sent.

**nIdn**: Identification number of the parameter to be written.

**nElement**: Element number of the parameter to be written. The following values are permitted:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Data status</td>
</tr>
<tr>
<td>0x02</td>
<td>Name (read only)</td>
</tr>
<tr>
<td>0x04</td>
<td>Attribute</td>
</tr>
<tr>
<td>0x08</td>
<td>Unit</td>
</tr>
<tr>
<td>0x10</td>
<td>Minimum</td>
</tr>
<tr>
<td>0x20</td>
<td>Maximum</td>
</tr>
<tr>
<td>0x40</td>
<td>Value</td>
</tr>
<tr>
<td>0x80</td>
<td>Default</td>
</tr>
</tbody>
</table>

**nDriveNo**: Drive number.

**bCommand**: This parameter must be set if internal command execution is to be used.

**pSrcBuf**: Address (pointer) to the send buffer.

**cbBufLen**: Number of date to be sent in bytes.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
END_VAR
```

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**nErrId**: Supplies the ADS error code associated with the most recently executed command if the bError output is set.

**Example of an implementation in ST:**

```plaintext
PROGRAM TEST_SoEWrite
  VAR
    fbSoeWrite : FB_EcSoEWrite;
    sNetId : T_AmsNetId := '172.16.2.131.2.1';
    bExecute : BOOL;
    nSlaveAddr : UINT := 1006;
  END_VAR
```

**Version:** 1.5
The **FB_SoERead_ByDriveRef** function block can be used to read drive parameters by means of the “Servo drive profile over EtherCAT (SoE)” protocol. To this end the slave must have a mailbox and support the SoE protocol. The drive parameter to be read is specified with the parameters nIdn (identification number), nElement and stDriveRef.

The global variable `bSeqReadDrvAttrAndValue := TRUE` from the Tc2_EtherCAT library can be used to enforce sequential access to attribute and value. The default value of this variable is FALSE. Devices of the AX5xxx series enable parallel and sequential access to attribute and value. For third-party devices it may be necessary to separate access to attribute and value, which overall slows down access by several cycles.

**VAR_INPUT**

```
VAR_INPUT
    stDriveRef : ST_DriveRef; (* contains sNetID of EcMaster, nSlaveAddr of EcDrive, nDriveNo of EcDrive, either preset or read from NC *)
    nIdn : WORD; (* SoE IDN: e.g. "S_0_IDN + 1" for S-0-0001 or "P_0_IDN + 23" for P-0-0023 *)
    nElement : BYTE; (* SoE element.*)
    pDstBuf : PVOID; (* Contains the address of the buffer for the received data. *)
    cbBufLen : UDINT; (* Contains the max. number of bytes to be received. *)
    bExecute : BOOL; (* Function block execution is triggered by a rising edge at this input.*)
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled.*)
END_VAR
```

**stDriveRef** The reference to the drive can be linked directly to the PLC in the System Manager. To this end an instance of `ST_PlcDriveRef` must be used and the NetID of the Byte array converted to a string. (type: `ST_DriveRef`)

**nIdn:** Identification number of the parameter to be read.

**nElement:** Element number of the parameter to be read. The following values are permitted:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Data status</td>
</tr>
<tr>
<td>0x02</td>
<td>Name (read only)</td>
</tr>
<tr>
<td>0x04</td>
<td>Attribute</td>
</tr>
<tr>
<td>0x08</td>
<td>Unit</td>
</tr>
<tr>
<td>0x10</td>
<td>Minimum</td>
</tr>
<tr>
<td>0x20</td>
<td>Maximum</td>
</tr>
<tr>
<td>0x40</td>
<td>Value</td>
</tr>
<tr>
<td>0x80</td>
<td>Default</td>
</tr>
</tbody>
</table>

**pDestBuf**: Address (pointer) to the read buffer.

**cbBufLen**: Maximum available buffer size (in bytes) for the data to be read.

**bExecute**: The function block is activated by a positive edge at this input.

**tTimeout**: Maximum time allowed for the execution of the function block.

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

**bBusy**: This output is set when the function block is activated, and remains set until an acknowledgement is received.

**bError**: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

**iAdsErrId**: In the case of a set bError output returns the ADS error code of the last executed command.

**iSercosErrId**: In the case of a set bError output returns the Sercos error of the last executed command.

**dwAttribute**: Returns the attributes of the Sercos parameter.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

#### 8.4 FB_SoEWrite_ByDriveRef

The function block `FB_SoEWrite_ByDriveRef` can be used to write drive parameters by means of the “Servo drive profile over EtherCAT (SoE)” protocol. To this end the slave must have a mailbox and support the SoE protocol. The drive parameter to be written is specified with the parameters nIdn (identification number), nElement and stDriveRef.
The global variable \texttt{bSeqReadDrvAttrAndValue := TRUE} from the \texttt{Tc2_EtherCAT} library can be used to enforce sequential access to attribute and value. The default value of this variable is \texttt{FALSE}. Devices of the AX5xxx series enable parallel and sequential access to attribute and value. For third-party devices it may be necessary to separate access to attribute and value, which overall slows down access by several cycles.

**VAR_INPUT**

\begin{verbatim}
VAR_INPUT
  stDriveRef : ST_DriveRef; (* contains sNetID of EcMaster, nSlaveAddr of EcDrive, nDriveNo of EcDrive, either preset or read from NC *)
  nIdn      : WORD; (* SoE IDN: e.g. "S_0_IDN + 1" for S-0-0001 or "P_0_IDN + 23" for P-0-0023*)
  nElement  : BYTE; (* SoE element.*)
  pSrcBuf   : PVOID; (* Contains the address of the buffer containing the data to be send. *)
  cbBufLen  : UDINT; (* Contains the max. number of bytes to be received. *)
  bExecute  : BOOL; (* Function block execution is triggered by a rising edge at this input.*)
  tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;(* States the time before the function is cancelled.*)
END_VAR
\end{verbatim}

\texttt{stDriveRef}: The reference to the drive can be linked directly to the PLC in the System Manager. To this end an instance of \texttt{ST_PlcDriveRef} must be used and the NetID of the Byte array converted to a string. (type: \texttt{ST_DriveRef})

\texttt{nIdn}: Identification number of the parameter to be read.

\texttt{nElement}: Element number of the parameter to be read. The following values are permitted:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Data status</td>
</tr>
<tr>
<td>0x02</td>
<td>Name (read only)</td>
</tr>
<tr>
<td>0x04</td>
<td>Attribute</td>
</tr>
<tr>
<td>0x08</td>
<td>Unit</td>
</tr>
<tr>
<td>0x10</td>
<td>Minimum</td>
</tr>
<tr>
<td>0x20</td>
<td>Maximum</td>
</tr>
<tr>
<td>0x40</td>
<td>Value</td>
</tr>
<tr>
<td>0x80</td>
<td>Default</td>
</tr>
</tbody>
</table>

\texttt{pSrcBuf}: Address (pointer) to the send buffer.

\texttt{cbBufLen}: Maximum available buffer size (in bytes) for the data to be read.

\texttt{bExecute}: The function block is activated by a positive edge at this input.

\texttt{tTimeout}: Maximum time allowed for the execution of the function block.

**VAR_OUTPUT**

\begin{verbatim}
VAR_OUTPUT
  bBusy      : BOOL;
  bError     : BOOL;
  iAdsErrId  : UINT;
  iSercosErrId : UINT;
END_VAR
\end{verbatim}

\texttt{bBusy}: This output is set when the function block is activated, and remains set until an acknowledgement is received.

\texttt{bError}: This output is set after the bBusy output has been reset when an error occurs in the transmission of the command.

\texttt{iAdsErrId}: In the case of a set bError output returns the ADS error code of the last executed command.

\texttt{iSercosErrId}: In the case of a set bError output returns the Sercos error of the last executed command:

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
9 Conversion Functions

9.1 F_ConvBK1120CouplerStateToString

The function F_ConvBK1120CouplerStateToString returns the coupler state of the BK1120/BK1150/BK1250 as string. For nState = 0 ‘No error’ is returned, otherwise ‘K-bus error’ is returned, e.g. for nState = 1. If several errors are pending, they are separated by commas.

VAR_INPUT

VAR_INPUT
nState : WORD;
END_VAR

nState: Coupler state; can be linked in the System Manager from the inputs of the BK1120/BK1250 to the PLC.

0x0000 = 'No error'
0x0001 = 'K-Bus error'
0x0002 = 'Configuration error'
0x0010 = 'Outputs disabled'
0x0020 = 'K-Bus overrun'
0x0040 = 'Communication error (Inputs)'
0x0080 = 'Communication error (Outputs)'

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

9.2 F_ConvMasterDevStateToString

The function F_ConvMasterDevStateToString converts the device status of the EtherCAT master to string.

For nState = 0 ‘OK’ is returned, otherwise, ‘Not OK – Link error’, e.g. for nState = 1. If several errors are pending, they are separated by hyphens.

VAR_INPUT

VAR_INPUT
nState : WORD;
END_VAR

nState: Device status of the EtherCAT master; can be linked as DevState in the System Manager from the inputs of the EtherCAT master to the PLC.

0x0001 = 'Link error'
0x0002 = 'I/O locked after link error (I/O reset required)'
0x0004 = 'Link error (redundancy adapter)'
0x0008 = 'Missing one frame (redundancy mode)'
0x0010 = 'Out of send resources (I/O reset required)'
0x0020 = 'Watchdog triggered'
0x0040 = 'Ethernet driver (miniport) not found'
0x0080 = 'I/O reset active'
0x0100 = 'At least one device in `INIT` state'
0x0200 = 'At least one device in `PRE-OP` state'
0x0400 = 'At least one device in `SAFE-OP` state'
0x0800 = 'At least one device indicates an error state'
0x1000 = 'DC not in sync'

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

9.3 **F_ConvProductCodeToString**

The function `F_ConvProductCodeToString` returns the product code as string, e.g. 'EL6731-0000-0017'. From version 3.3.8.0 of the Tc2_EtherCAT library this function also supports ELM and EPP slaves such as 'EPP4374-0002-0018' and 'ELM3704-0001-0016'.

VAR_INPUT

```plaintext
VAR_INPUT
   stSlaveIdentity : ST_EcSlaveIdentity;
END_VAR
```

`stSlaveIdentity`: Slave identity, which can be read with `FB_EcGetSlaveIdentity[28]`.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

9.4 **F_ConvSlaveStateToString**

The function `F_ConvSlaveStateToString` returns the EtherCAT slave state as string. For conversion to the string see `F_ConvStateToString[68]`.

VAR_INPUT

```plaintext
VAR_INPUT
   state : ST_EcSlaveState;
END_VAR
```

`state`: EtherCAT slave state structure (consisting of: deviceState : BYTE; linkState : BYTE;)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
### 9.5 F_ConvSlaveStateToBits

The function `F_ConvSlaveStateToBits` returns the EtherCAT slave state as structure `TYPE ST_EcSlaveStateBits [105].

**VAR_INPUT**

```plaintext
VAR_INPUT
    stEcSlaveState : ST_EcSlaveState;
END_VAR
```

`stEcSlaveState`: EtherCAT slave state structure (consisting of: deviceState : BYTE; linkState : BYTE;)

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 9.6 F_ConvSlaveStateToBitsEx

The function `F_ConvSlaveStateToBitsEx` returns the EtherCAT slave state as structure `ST_EcSlaveStateBitsEx [106].

**VAR_INPUT**

```plaintext
VAR_INPUT
    stEcSlaveState : ST_EcSlaveState;
END_VAR
```

`stEcSlaveState`: EtherCAT slave state structure (consisting of: deviceState : BYTE; linkState : BYTE;)

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 9.7 F_ConvStateToString

The function `F_ConvStateToString` returns the EtherCAT slave state as string. For `nState = 0` ’ ’ is returned, otherwise, ‘INIT ’ is returned, e.g. for `nState = 1`. If several messages are pending, they are separated by spaces.
VAR_INPUT
  nState : WORD;
END_VAR

nState: EtherCAT slave state as WORD

0x__1 = 'INIT'
0x__2 = 'PREOP'
0x__3 = 'BOOT'
0x__4 = 'SAFEOP'
0x__8 = 'OP'
0x001_ = 'Slave signals error'
0x002_ = 'Invalid vendorId, productCode... read'
0x004_ = 'Initialization error occurred'
0x008_ = 'Slave disabled'
0x010_ = 'Slave not present'
0x020_ = 'Slave signals link error'
0x040_ = 'Slave signals missing link'
0x080_ = 'Slave Slave signals unexpected link'
0x100_ = 'Communication port A'
0x200_ = 'Communication port B'
0x400_ = 'Communication port C'
0x800_ = 'Communication port D'

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10 Distributed Clocks

10.1 DCTIME32

10.1.1 ConvertDcTimeToPos

This function block converts a 32-bit distributed clock system time variable of type T_DCTIME32 (108) to a corresponding NC axis position (i.e. the NC axis position at precisely this time).

VAR_INPUT

VAR_INPUT
nAxisId : UDINT;
nSubIdx : UDINT;
dcTime : T_DCTIME32;(* 32 bit distributed clock time *)
END_VAR

nAxisId: ID of the NC axis

nSubIdx: This 32-bit input variable contains two different items of information, and is therefore divided into two 16-bit values:

• The low word (the 16 bits with the lowest value) contains the subindex for relative addressing of an encoder subelement at an axis. The subindex is counted upwards from zero. For the typical case of an axis that has just one encoder, the null subindex is correct.

• The high word (the 16 bits with the highest value) contains a control word (bit mask) that affects the way in which the position is calculated (e.g. the type of interpolation or extrapolation). A bit mask value of 0x0001 means that the set acceleration of the axis is to be included in the calculation.

dcTime: 32-bit distributed clock system time variable. This input magnitude is converted through calculation into the corresponding NC axis position.

NOTE

The 32-bit time may only be used in the narrow range of ± 2,147 seconds around the current system time, to ensure that it is unambiguous. Within the function block this prerequisite cannot be checked.

VAR_OUTPUT

VAR_OUTPUT
fPosition : LREAL;
iErr : UDINT;
END_VAR

fPosition: Supplies the NC axis position corresponding to dcTime. This is an NC axis position that has been scaled and provided with an offset, having, for instance, physical units of degrees or of millimetres.

iErr: Returns the error number if an error occurs, e.g. error 0x4012 (axis ID is not allowed, or axis does not exist within the system).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.1.2 ConvertPosToDcTime

This function block converts an NC axis position to a corresponding 32-bit distributed clock system time variable of type T_DCTIME32 [108] (i.e. the time which precisely this NC axis position was or will be reached).

VAR_INPUT

VAR_INPUT

nAxisId : UDINT;
nSubIdx : UDINT;
fPosition : LREAL;
END_VAR

nAxisId: ID of the NC axis.

nSubIdx: This 32-bit input magnitude is composed of two different items of information, and is divided into two 16-bit values:

- The low word (the 16 bits with the lowest value) contains the subindex for relative addressing of an encoder subelement at an axis. The subindex is counted upwards from zero. For the typical case of an axis that has just one encoder, the null subindex is correct.
- The high word (the 16 bits with the highest value) contains a control word (bit mask) that affects the way in which the position is calculated (e.g. the type of interpolation or extrapolation). A bit mask value of 0x0001 means that the set acceleration of the axis is to be included in the calculation.

fPosition: NC axis position that is converted to the corresponding 32-bit distributed clock system time variable.
If the distributed clock system time that corresponds to the position is outside the expected time window of ±2,147 seconds, this conversion is rejected with an error number.

VAR_OUTPUT

VAR_OUTPUT

dcTime : T_DCTIME32;(* 32 bit distributed clock time *)
iErr : UDINT;
END_VAR

dcTime: Returns the 32-bit distributed clock system time variable that corresponds to input fPosition.
iErr: Supplies an error number if an error occurs, e.g.

- Error 0x4012: axis ID is not allowed, or axis is not present in the system,
- Error 0x4361: time window exceeded (future),
- Error 0x4362: time window exceeded (past),
- Error 0x4363: position cannot be found mathematically.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.1.3 ConvertDcTimeToPathPos

This function block converts a 32-bit distributed clock system time variable of type T_DCTIME32 to a relative Nci path distance on the contour of the currently active Nci program (i.e. the function block returns a positive or negative relative interval, depending on the timing).

**VAR_INPUT**

```plaintext
VAR_INPUT
    nGrpId : UDINT;
    nSubIdx : UDINT;
    dcTime : T_DCTIME32; (* 32 bit distributed clock time *)
END_VAR
```

- **nGrpId**: Group ID of the corresponding Nci channel
- **nSubIdx**: This 32-bit input variable contains two different items of information, and is therefore divided into two 16-bit values:
  - The low word (the 16 bits with the lowest value) contains the subindex for relative addressing of an encoder subelement at an axis. The subindex is counted upwards from zero. For the typical case of an axis that has just one encoder, the null subindex is correct.
  - The high word (the 16 bits with the highest value) contains a control word (bit mask) that affects the way in which the position is calculated (e.g. the type of interpolation or extrapolation). A bit mask value of 0x0001 means that the set acceleration of the axis is to be included in the calculation. Bit mask 0x0010 means that the calculation is relative and currently compulsory. Otherwise the call is rejected with an error.
- **dcTime**: 32-bit distributed clock system time variable. This input variable is converted to the corresponding relative Nci path distance on the contour.

**NOTE**

The 32-bit time may only be used in the narrow range of ± 2,147 seconds around the current system time, to ensure that it is unambiguous. Within the function block this prerequisite cannot be checked.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
    fPosition : LREAL;
    iErr : UDINT;
END_VAR
```

- **fPosition**: Returns the relative Nci path distance on the contour that corresponds to the dcTime.
- **iErr**: Returns an error number in the event of an error.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.1.4 ConvertPathPosToDcTime

This function block converts a relative Nci path distance to a corresponding 32-bit distributed clock system time variable of type `T_DCTIME32` (i.e. the time that corresponds or corresponded to the relative Nci path distance).

**VAR_INPUT**

```plaintext
VAR_INPUT
    nGrpId : UDINT;
    nSubIdx : UDINT;
    fPosition : LREAL;
END_VAR
```

**nGrpId**: Group ID of the corresponding Nci channel

**nSubIdx**: This 32-bit input variable contains two different items of information, and is therefore divided into two 16-bit values:

- The low word (the 16 bits with the lowest value) contains the subindex for relative addressing of an encoder subelement at an axis. The subindex is counted upwards from zero. For the typical case of an axis that has just one encoder, the null subindex is correct.
- The high word (the 16 bits with the highest value) contains a control word (bit mask) that affects the way in which the position is calculated (e.g. the type of interpolation or extrapolation).
  A bit mask value of 0x0001 means that the set acceleration of the axis is to be included in the calculation.
  Bit mask 0x0010 means that the calculation is relative and currently compulsory. Otherwise the call is rejected with an error.

**fPosition**: Relative Nci path distance, which is converted to the corresponding 32-bit distributed clock system time.
If the distributed clock system time that corresponds to the relative Nci path distance is outside the expected time window of ±2,147 seconds, this conversion is rejected with an error number.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
    dcTime : T_DCTIME32; (* 32 bit distributed clock time *)
    iErr : UDINT;
END_VAR
```

**dcTime**: Returns the 32-bit distributed clock system time variable that corresponds to input fPosition.

**iErr**: Supplies an error number if an error occurs, e.g.

- Error 0x4361: time window exceeded (future),
- Error 0x4362: time window exceeded (past),
- Error 0x4363: position cannot be found mathematically.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.2 DCTIME64

10.2.1 DCTIME_TO_DCTIME64

The function converts a distributed clock system time variable of type \( T\text{\_DCTIME} \) to a 64-bit distributed clock system time variable of type \( T\text{\_DCTIME64} \).

FUNCTION DCTIME_TO_DCTIME64: T\_DCTIME64

VAR_INPUT
  in : T\_DCTIME;
END_VAR

in: The distributed clock system time variable to be converted

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.2.2 DCTIME64_TO_DCTIME

The function converts a distributed clock system time variable of type \( T\text{\_DCTIME} \) to a 64-bit distributed clock system time variable of type \( T\text{\_DCTIME64} \).

FUNCTION DCTIME64_TO_DCTIME: T\_DCTIME

VAR_INPUT
  in : T\_DCTIME64;
END_VAR

in: The distributed clock system time variable to be converted

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.2.3 DCTIME64_TO_DCTIMESTRUCT

The function converts a 64-bit distributed clock system time variable of type \( T\text{\_DCTIME64} \) to a structured variable of type \( \text{DCTIMESTRUCT} \).

FUNCTION DCTIME64_TO_DCTIMESTRUCT

VAR_INPUT
  in : T\_DCTIME64;
END_VAR
in: The distributed clock system time variable to be converted

Example:

```
PROGRAM P_TEST
VAR
  dcStruct : DCTIMESTRUCT;
  dcTime : T_DCTIME64;
END_VAR

dcTime := F_GetCurDcTickTime64();
dcStruct := DCTIME64_TO_DCTIMESTRUCT (dcTIME);
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.2.4 DCTIME64_TO_FILETIME64

The function converts a 64-bit "Distributed Clock System Time" variable of type T_DCTIME64 to a 64-bit "Windows File Time" variable of type T_FILETIME64.

```
FUNCTION DCTIME64_TO_FILETIME64: T_FILETIME64
VAR_INPUT
  in : T_DCTIME64;
END_VAR;

in: the "Distributed Clock System Time" variable to be converted

Sample:
```
PROGRAM P_TEST
VAR
  ft := T_FILETIME64;
  dct : T_DCTIME64;
END_VAR

dct := F_GetCurDcTickTime64();
ft := DCTIME64_TO_FILETIME64(dct);
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT &gt;= 3.3.16.0</td>
</tr>
</tbody>
</table>

10.2.5 DCTIME64_TO_STRING

The function converts a 64-bit distributed clock system time variable of type T_DCTIME64 to a string.

The string resulting the conversion has the following format: ‘YYYY-MM-DD-hh:mm:ss.nnnnnnnnnn’

- YYYY: year;
- MM: month;
- DD: day;
Distributed Clocks

- hh: hour;
- mm: minute;
- ss: second;
- nnnnnnnn: nanoseconds

**FUNCTION DCTIME64_TO_STRING: STRING (29)**

```plaintext
VAR_INPUT
  in : T_DCTIME64; (*Distributed Clock Time*)
END_VAR

in: The distributed clock system time variable to be converted

Example:
See description of the function F_GetCurDcTickTime64 [82].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.2.6 DCTIME64_TO_SYSTEMTIME**

The function converts a 64-bit distributed clock system time variable of type T_DCTIME64 to a structured Windows system time variable of type TIMESTRUCT.

**DCTIME64_TO_SYSTEMTIME: TIMESTRUCT**

```plaintext
VAR_INPUT
  in : T_DCTIME64;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

```plaintext
PROGRAM P_TEST
VAR
  syst : TIMESTRUCT
END_VAR

syst := DCTIME64_TO_SYSTEMTIME ( F_GetCurDcTickTime64() )
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.2.7 DCTIMESTRUCT_TO_DCTIME64**

The function converts a structured variable of type DCTIMESTRUCT to a 64-bit distributed clock system time variable T_DCTIME64. The structure components wDayOfWeek is ignored in the conversion. The structure components wYear must be greater than or equal to 2000 and less than 2584. For invalid values of the structure components the function returns the value zero.
FUNCTION DCTIMESTRUCT_TO_DCTIME64: T_DCTIME64

VAR_INPUT
  in : DCTIMESTRUCT;
END_VAR

in: The structured variable to be converted

Example:

PROGRAM P_TEST
VAR
  dcStruct : DCTIMESTRUCT := ( wYear := 2008, wMonth := 3, wDay := 13,
                             wHour := 1, wMinute := 2, wSecond := 3,
                             wMilliseconds := 123, wMicroseconds := 456, wNanoseconds := 789 );
  dc64 : T_DCTIME64;
END_VAR

dc64 := DCTIMESTRUCT_TO_DCTIME64( dcStruct );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.2.8 FILETIME64_TO_DCTIME64

The function converts a 64-bit "Windows File Time" variable of type T_FILETIME64 to a 64-bit "Distributed Clock System Time" variable of type T_DCTIME64. In the event of a conversion error the function returns the value zero.

FUNCTION FILETIME64_TO_DCTIME64: T_DCTIME64

VAR_INPUT
  in : T_FILETIME64;
END_VAR

in: The "Windows File Time" variable to be converted.

Sample:

PROGRAM P_TEST
VAR
  ft : T_FILETIME64;
  dct : T_DCTIME64;
END_VAR

ft := F_GetSystemTime();
dct := FILETIME64_TO_DCTIME64( ft );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT &gt;= 3.3.16.0</td>
</tr>
</tbody>
</table>

10.2.9 STRING_TO_DCTIME64

The function converts a string to a distributed clock system time variable of type T_DCTIME64.
FUNCTION STRING_TO_DCTIME64: T_DCTIME64

VAR_INPUT
  in : STRING(29);
END_VAR

in: The string to be converted.

The string must have the following format: ‘YYYY-MM-DD-hh:mm:ss.nnnnnnnnn’

- YYYY: year;
- MM: month;
- DD: day;
- hh: hour;
- mm: minute;
- ss: second;
- nnnnnnnn: nanoseconds

Example:

See description of the function F_GetCurDcTickTime64 [82].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2EtherCAT</td>
</tr>
</tbody>
</table>

10.2.10 SYSTEMTIME_TO_DCTIME64

The function converts a structured Windows system time variable of type TIMESTRUCT to a 64-bit distributed clock system time variable of type T_DCTIME64 [108]. In the event of a conversion error the function returns the value zero.

FUNCTION SYSTEMTIME_TO_DCTIME64: T_DCTIME64

VAR_INPUT
  in : TIMESTRUCT;
  micro : WORD(0..999); (* Microseconds: 0..999 *)
  nano : WORD(0..999); (* Nanoseconds: 0..999 *)
END_VAR

in: The Windows system time variable to be converted

micro: microseconds

nano: nanoseconds

Example:

PROGRAM P_TEST

VAR
  syst : TIMESTRUCT := ( wYear := 2009, wMonth := 9, wDay := 16, wHour := 12, wMinute := 22, wSecond := 44, wMilliseconds := 123 );
END_VAR

dct := SYSTEMTIME_TO_DCTIME64( syst, 456, 789 );
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.2.11 FB_EcDcTimeCtrl64

This function block can be used to read the individual components such as year, month, day etc. of a 64-bit distributed clock system time variable of type T_DCTIME64 [108]. The function block has several A_GETXYZ actions. Once the required action has been called, the value of the XYZ component is available in the “get” output variable. The “put” input variable is currently not used.

The function block features the following tasks:

- A_GetYear
- A_GetMonth
- A_GetDay
- A_GetDayOfWeek
- A_GetHour
- A_GetMinute
- A_GetSecond
- A_GetMilli
- A_GetNano

**VAR_IN_OUT**

```
VAR_IN_OUT
  in : T_DCTIME64;
END_VAR
```

*in*: TwinCAT distributed clock system time variable

**VAR_INPUT**

```
VAR_INPUT
  put : WORD;
END_VAR
```

*put*: Input parameter (currently not used)

**VAR_OUTPUT**

```
VAR_OUTPUT
  bError : BOOL;
  get    : WORD;
END_VAR
```

*bError*: This output is set if an error has occurred during the action call.

*get*: Output parameter (year, month, day, etc.)

Example of an implementation in ST:

```
PROGRAM P.TEST
VAR
  dcStruct : DCTIMESTRUCT;
  dcTime   : T_DCTIME64;
  fbCtrl   : FB_EcDcTimeCtrl;
```
Distributed Clocks

wYear : WORD;
wMonth : WORD;
wDay : WORD;
wDayOfWeek : WORD;
wHour : WORD;
wMinute : WORD;
wSecond : WORD;
wMilli : WORD;
wMicro : WORD;
wNano : WORD;
END_VAR

dcTime := F_GetCurDcTickTime64();
fbCtrl.A_GetYear( in := dcTime, get => wYear );
fbCtrl.A_GetMonth( in := dcTime, get => wMonth );
fbCtrl.A_GetDay( in := dcTime, get => wDay );
fbCtrl.A_GetDayOfWeek( in := dcTime, get => wDayOfWeek );
fbCtrl.A_GetHour( in := dcTime, get => wHour );
fbCtrl.A_GetMinute( in := dcTime, get => wMinute );
fbCtrl.A_GetSecond( in := dcTime, get => wSecond );
fbCtrl.A_GetMilli( in := dcTime, get => wMilli );
fbCtrl.A_GetMicro( in := dcTime, get => wMicro );
fbCtrl.A_GetNano( in := dcTime, get => wNano );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3 DCTIME64 and ULINT

10.3.1 F_ConvExtTimeToDcTime64

The function F_ConvExtTimeToDcTime64 converts an external time to the TwinCAT distributed clock system time.

FUNCTION F_ConvExtTimeToDcTime64: T_DCTIME64

VAR_INPUT

ExtTime : T_DCTIME64;
DcToExtTimeOffset : ULINT;
END_VAR

ExtTime: External time in TwinCAT distributed clock system time format
DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.2 F_ConvTcTimeToDcTime64

The function F_ConvTcTimeToDcTime64 converts the TwinCAT system time to the TwinCAT distributed clock system time.
FUNCTION F_ConvTcTimeToDcTime64: T_DCTIME64

VAR_INPUT
  TcTime : T_DCTIME64;
  DcToTcTimeOffset : ULLINT;
END_VAR

TcTime: TwinCAT system time in TwinCAT distributed clock system time format

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.3 F_ConvTcTimeToExtTime64

The function F_ConvTcTimeToExtTime64 converts the TwinCAT distributed clock system time to an external time.

FUNCTION F_ConvTcTimeToExtTime64: T_DCTIME64

VAR_INPUT
  TcTime : T_DCTIME64;
  DcToTcTimeOffset : ULLINT;
  DcToExtTimeOffset : ULLINT;
END_VAR

TcTime: TwinCAT system time in distributed clock format

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.4 F_GetActualDcTime64

This function returns the current time in TwinCAT distributed clock system time format (T_DCTIME64 [108]).

FUNCTION F_GetActualDcTime: T_DCTIME64

VAR_INPUT
  (*none*)
END_VAR

Sample in ST:
Distributed Clocks

PROGRAM MAIN
VAR
    actDC : T_DCTIME64;
    sAct  : STRING;
END_VAR

actDC := F_GetActualDcTime64();
sAct  := DCTIME64_TO_STRING( actDC );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.5  F_GetCurDcTaskTime64

This function returns the task start time (time at which the task should start) in TwinCAT distributed clock system time format (T_DCTIME64). The function always returns the start time of the task in which it was called.

FUNCTION F_GetCurDcTaskTime64: T_DCTIME64

VAR_INPUT
    (*none*)
END_VAR

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.6  F_GetCurDcTickTime64

The function returns the time of the current (last) tick in TwinCAT distributed clock system time format (T_DCTIME64).

FUNCTION F_GetCurDcTickTime64: T_DCTIME64

VAR_INPUT
    (*none*)
END_VAR

Example:

PROGRAM MAIN
VAR
    tDC : T_DCTIME64;
    sDC : STRING;
    tDCBack : T_DCTIME64;

    sDCZero : STRING; (* DCTIME64 = zero time starts on 01.01.2000 *)
    tDCBackZero : T_DCTIME64;

    tDCFromString : T_DCTIME64;
    sDCBackFromString : STRING;
END_VAR

tDC := F_GetCurDcTickTime64();
sDC := DCTIME64_TO_STRING( tDC );
tDCback := STRING_TO_DCTIME64( sDC );

sDCZero := DCTIME64_TO_STRING( UINTEGER(0, 0) );
tDCbackFromZero := STRING_TO_DCTIME64( sDCZero );

tDCFromString := STRING_TO_DCTIME64( '2007-03-09-11:31:09.223456789' );
sDCBackFromString := DCTIME64_TO_STRING( tDCFromString );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.7  F_GetCurExtTime64

The function returns the external time in TwinCAT distributed clock system time format (T_DCTIME64).

FUNCTION F_GetCurExtTime64: T_DCTIME64

VAR_INPUT

DcToExtTimeOffset : ULINT;
DcToTcTimeOffset : ULINT;
END_VAR

DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.3.8  FB_EcExtSyncCalcTimeDiff64

The function block FB_EcExtSyncCalcTimeDiff64 calculates the difference between external and internal time, taking into account the time offsets.

VAR_IN_OUT

VAR_IN_OUT

DcToTcTimeOffset : ULINT;
DcToExtTimeOffset : ULINT;
ExtTime : T_DCTIME64;
IntTime : T_DCTIME64;
END_VAR

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time
**ExtTime**: External time in TwinCAT distributed clock system time format

**IntTime**: Internal time in TwinCAT distributed clock system time format

**VAR_OUTPUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nTimeDiff</td>
<td>UDINT; (<em>with difference greater than 32 bit timeDiff = 0xffffffff</em>)</td>
</tr>
<tr>
<td>nOffsetFromSyncMaster</td>
<td>DINT; (<em>less than 32 bit int Offset = 0x80000000, greater than 32 bit int Offset = 0x7FFFFFFF</em>)</td>
</tr>
</tbody>
</table>

- **nTimeDiff**: If the difference is less than 32 bit, the time difference is returned. If the difference is greater than 32 bit, 0xffffffff is returned.
- **nOffsetFromSyncMaster**: If the difference is greater than 32 bit and the offset between internal and DC time is less than 32 bit, 0x80000000 is returned. If the difference is greater than 32 bit and the offset between internal and DC time is greater than 32 bit, 0x7FFFFFFF is returned.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.3.9 FB_EcExtSyncCheck64**

The function block **FB_EcExtSyncCheck64** checks whether the internal and external clocks are synchronous. See function block **FB_EcExtSyncCalcTimeDiff64** [83].

**VAR_INPUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nSyncWindow</td>
<td>UDINT;</td>
</tr>
<tr>
<td>bNotConnected</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

- **nSyncWindow**: Time window within which the internal and external clock are regarded as synchronous.
- **bNotConnected**: TRUE = connection to external clock is interrupted.

**VAR_IN_OUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DcToTcTimeOffset</td>
<td>T_LARGE_INTEGER;</td>
</tr>
<tr>
<td>DcToExtTimeOffset</td>
<td>T_LARGE_INTEGER;</td>
</tr>
<tr>
<td>ExtTime</td>
<td>T_DCTIME64;</td>
</tr>
<tr>
<td>IntTime</td>
<td>T_DCTIME64;</td>
</tr>
</tbody>
</table>

- **DcToTcTimeOffset**: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time
- **DcToExtTimeOffset**: Time offset between the TwinCAT distributed clock system time and an external time
- **ExtTime**: External time in TwinCAT distributed clock system time format
- **IntTime**: Internal time in TwinCAT distributed clock system time format
VAR_OUTPUT

VAR_OUTPUT
  bSynchronized     : BOOL;
  nTimeDiff         : UDINT;
  nOffsetFromSyncMaster : DINT;
END_VAR

bSynchronized: TRUE = external and internal clock are synchronous
nTimeDiff: Current time difference between the two clocks
nOffsetFromSyncMaster: Offset to sync master

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4 [obsolete]

10.4.1 [outdated DCTIME]

10.4.1.1 DCTIME_TO_DCTIMESTRUCT

FUNCTION DCTIME_TO_DCTIMESTRUCT: DCTIMESTRUCT

VAR_INPUT
  in : T_DCTIME;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

PROGRAM P_TEST
VAR
  dcStruct : DCTIMESTRUCT;
  dcTime : T_DCTIME;
END_VAR

dcTime := F_GetCurDcTickTime();
dcStruct := DCTIME_TO_DCTIMESTRUCT(dcTime);

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.4.1.2 DCTIME_TO_FILETIME

Outdated function
This function is outdated. Use the function DCTIME64_TO_FILETIME [97] instead.

The function converts a 64-bit distributed clock system time variable of type T_DCTIME [109] to a 64-bit Windows file time variable of type T_FILETIME.

FUNCTION DCTIME_TO_FILETIME: T_FILETIME
VAR_INPUT
  in : T_DCTIME;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

PROGRAM P_TEST
VAR
  ft : T_FILETIME;
  dct : T_DCTIME;
END_VAR

dct := F_GetCurDcTickTime();
ft := DCTIME_TO_FILETIME(dct);

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.1.3 DCTIME_TO_STRING

Outdated function
This function is outdated. Use the function DCTIME64_TO_STRING [75] instead.

The function converts a string to a distributed clock system time variable of type T_DCTIME [109].

The string resulting the conversion has the following format: "YYYY-MM-DD-hh:mm:ss.nnnnnnnnnn"

- YYYY: year;
- MM: month;
- DD: day;
- hh: hour;
- mm: minute;
- ss: second;
- nnnnnnnnnn: nanoseconds;
FUNCTION DCTIME_TO_STRING: STRING(29)

VAR_INPUT
  in : T_DCTIME;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

See description of the function \F_GetCurDcTickTime\ [\ref 94].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.1.4 DCTIME_TO_SYSTEMTIME

The function converts a 64-bit distributed clock system time variable of type T_DCTIME [\ref 109] to a structured Windows system time variable of type TIMESTRUCT.

FUNCTION DCTIME_TO_SYSTEMTIME: TIMESTRUCT

VAR_INPUT
  in : T_DCTIME;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

PROGRAM P_TEST
VAR
  syst : TIMESTRUCT;
END_VAR

syst := DCTIME_TO_SYSTEMTIME( F_GetCurDcTickTime() );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.1.5 DCTIMESTRUCT_TO_DCTIME

The function is outdated. Use the function DCTIME64_TO_SYSTEMTIME [\ref 76] instead.

FUNCTION DCTIMESTRUCT_TO_DCTIME: T_DCTIME

VAR_INPUT
  in : TIMESTRUCT;
END_VAR

in: The distributed clock system time variable to be converted.

Example:

PROGRAM P_TEST
VAR
  syst : TIMESTRUCT;
END_VAR

syst := DCTIMESTRUCT_TO_DCTIME(  );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
The function converts a structured variable of type DCTIMESTRUCT to a 64-bit distributed clock system time variable of type T_DCTIME.
The structure components wDayofWeek is ignored in the conversion. The structure components wYear must be greater than or equal to 2000 and less than 2584. For invalid values of the structure components the function returns the value zero.

**FUNCTION DCTIMESTRUCT_TO_DCTIME: T_DCTIME**

```plaintext
VAR_INPUT
   in : DCTIMESTRUCT;
END_VAR

in: The structured variable to be converted.

Example:

PROGRAM P_TEST
VAR
   dcStruct : DCTIMESTRUCT := ( wYear := 2008, wMonth := 3, wDay := 13,
                              wHour := 1, wMinute := 2, wSecond :=3,
                              wMilliseconds := 123, wMicroseconds := 456, wNanoseconds := 789 );
   dc64 : T_DCTIME;
END_VAR

dc64 := DCTIMESTRUCT_TO_DCTIME( dcStruct );
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.4.1.6 FILETIME_TO_DCTIME**

The function converts a 64-bit Windows file time variable of type T_FILETIME to a 64-bit distributed clock system time variable of type T_DCTIME. In the event of a conversion error the function returns the value zero.

**FUNCTION FILETIME_TO_DCTIME: T_DCTIME**

```plaintext
VAR_INPUT
   in : T_FILETIME;
END_VAR

in: The Windows file time variable to be converted.

Example:

PROGRAM P_TEST
VAR
   fbSysFileTime : GETSYSTEMTIME;
   ft : T_FILETIME;
   dct : T_DCTIME;
END_VAR

fbSysFileTime(timeLoDW=>ft.dwLowDateTime, timeHiDW=>ft.dwHighDateTime);

dct := FILETIME_TO_DCTIME(ft);
```
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.1.7 STRING_TO_DCTIME

**Outdated function**

This function is outdated. Use the function STRING_TO_DCTIME64 [77] instead.

The function converts a string to a distributed clock system time variable of type T_DCTIME [109].

**FUNCTION STRING_TO_DCTIME: T_DCTIME**

```plaintext
FUNCTION STRING_TO_DCTIME: T_DCTIME
VAR_INPUT
    in : STRING(29);
END_VAR

in: The string to be converted.

The string must have the following format: 'YYYY-MM-DD-hh:mm:ss.nnnnnnnnnn'

- YYYY: year;
- MM: month;
- DD: day;
- hh: hour;
- mm: minute;
- ss: second;
- nnnnnnnnn: nanoseconds;

Example:

See description of the function F_GetCurDcTickTime [94].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.1.8 SYSTEMTIME_TO_DCTIME

**Outdated function**

This function is outdated. Use the function SYSTEMTIME_TO_DCTIME64 [78] instead.
The function converts a structured Windows system time variable of type TIMESTRUCT to a 64-bit distributed clock system time variable of type T_DCTIME \(^{[109]}\). In the event of a conversion error the function returns the value zero.

**FUNCTION SYSTEMTIME_TO_DCTIME: T_DCTIME**

VAR_INPUT

| in: TIMESTRUCT; |
| micro : WORD(0..999); (* Microseconds: 0..999 *) |
| nano : WORD(0..999); (* Nanoseconds: 0..999 *) |

END_VAR

in: The Windows system time variable to be converted.

micro: Microseconds.

nano: Nanoseconds.

**Example:**

```plaintext
PROGRAM P_TEST
VAR
  syst : TIMESTRUCT := (wYear := 2009, wMonth := 9, wDay := 16, wHour := 12, wMinute := 22, wSecond := 44, wMilliseconds := 123);
END_VAR

dct := SYSTEMTIME_TO_DCTIME(syst, 456, 789);
```

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.4.1.9 FB_EcDcTimeCtrl**

- **Outdated function**

This function is outdated. Use the function block **FB_EcDcTimeCtrl64** \(^{[79]}\) instead.

This function block can be used to read the individual components such as year, month, day etc. of a 64-bit distributed clock system time variable of type T_DCTIME \(^{[109]}\). The function block has several A_GetXYZ actions. Once the required action has been called, the value of the XYZ component is available in the “get” output variable. The “put” input variable is currently not used.

The function block currently has the following actions:

- A_GetYear;
- A_GetMonth;
- A_GetDay;
- A_GetDayOfWeek;
- A_GetHour;
- A_GetMinute;
- A_GetSecond;
- A_GetMilli;
- A_GetMicro;
- A_GetNano;
VAR_IN_OUT
VAR_IN_OUT
  in : T_DCTIME;
END_VAR

in: TwinCAT distributed clock system time variable

VAR_INPUT
VAR_INPUT
  put : WORD;
END_VAR

put: Input parameter (currently not used)

VAR_OUTPUT
VAR_OUTPUT
  bError : BOOL;
  get : WORD;
END_VAR

bError: This output is set if an error has occurred during the action call.

get: Output parameter (year, month, day, etc.)

Example of an implementation in ST:

PROGRAM P_TEST
VAR
  dcStruct : DCTIMESTRUCT;
  dcTime : T_DCTIME;
  fbCtrl : FB_EcDcTimeCtrl;
  wYear : WORD;
  wMonth : WORD;
  wDay : WORD;
  wDayOfWeek : WORD;
  wHour : WORD;
  wMinute : WORD;
  wSecond : WORD;
  wMilli : WORD;
  wMicro :WORD;
  wNano : WORD;
END_VAR

  dcTime := F_GetCurDcTickTime();
  fbCtrl.A_GetYear( in := dcTime, get => wYear );
  fbCtrl.A_GetMonth( in := dcTime, get => wMonth );
  fbCtrl.A_GetDay( in := dcTime, get => wDay );
  fbCtrl.A_GetDayOfWeek( in := dcTime, get => wDayOfWeek );
  fbCtrl.A_GetHour( in := dcTime, get => wHour );
  fbCtrl.A_GetMinute( in := dcTime, get => wMinute );
  fbCtrl.A_GetSecond( in := dcTime, get => wSecond );
  fbCtrl.A_GetMilli( in := dcTime, get => wMilli );
  fbCtrl.A_GetMicro( in := dcTime, get => wMicro );
  fbCtrl.A_GetNano( in := dcTime, get => wNano );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.4.2 [outdated DCTIME and T_LARGE_INTEGER]

10.4.2.1 F_ConvExtTimeToDcTime

The function F_ConvExtTimeToDcTime converts an external time to the TwinCAT distributed clock system time.

FUNCTION F_ConvExtTimeToDcTime: T_DCTIME
VAR_INPUT
  ExtTime : T_DCTIME;
  DcToExtTimeOffset : T_LARGE_INTEGER;
END_VAR

ExtTime: External time in TwinCAT distributed clock system time format
DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.2.2 F_ConvTcTimeToDcTime

The function F_ConvTcTimeToDcTime64 converts the TwinCAT system time to the TwinCAT distributed clock system time.

FUNCTION F_ConvTcTimeToDcTime: T_DCTIME
VAR_INPUT
  TcTime : T_DCTIME;
  DcToTcTimeOffset : T_LARGE_INTEGER;
END_VAR

TcTime: TwinCAT system time in TwinCAT distributed clock system time format
DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
10.4.2.3  

**F_ConvTcTimeToExtTime**

**Outdated function**

This function is outdated. Use the function `F_ConvTcTimeToExtTime64` instead.

The function `F_ConvTcTimeToExtTime` converts the TwinCAT distributed clock system time to an external time.

**FUNCTION F_ConvTcTimeToExtTime: T_DCTIME**

```
VAR_INPUT
   TcTime            :  T_DCTIME;
   DcToTcTimeOffset  :  T_LARGE_INTEGER;
   DcToExtTimeOffset :  T_LARGE_INTEGER;
END_VAR
```

**TcTime**: TwinCAT system time in distributed clock format

**DcToTcTimeOffset**: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

**DcToExtTimeOffset**: Time offset between the TwinCAT distributed clock system time and an external time

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.2.4  

**F_GetActualDcTime**

**Outdated function**

This function is outdated. Use the function `F_GetActualDcTime64` instead.

This function returns the current time in TwinCAT distributed clock system time format (T_DCTIME).

**FUNCTION F_GetActualDcTime: T_DCTIME**

```
VAR_INPUT
    (*none*)
END_VAR
```

**Example:**

```
PROGRAM MAIN
VAR
    actDC : T_DCTIME;
    sAct : STRING;
END_VAR

actDC := F_GetActualDcTime();
sAct := DCTIME_TO_STRING( actDC );
```
Distributed Clocks

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 10.4.2.5 \(F_{\text{GetCurDcTaskTime}}\)

**Outdated function**

This function is outdated. Use the function \(F_{\text{GetCurDcTaskTime64}}\) instead.

This function returns the task start time (time at which the task should start) in TwinCAT distributed clock system time format \((T\_DCTIME)\). The function always returns the start time of the task in which it was called.

**FUNCTION** \(F_{\text{GetCurDcTaskTime}}: T\_DCTIME\)

**VAR_INPUT**

\[\text{(*none*)}\]

**END_VAR**

### 10.4.2.6 \(F_{\text{GetCurDcTickTime}}\)

**Outdated function**

The function is outdated. Use the function \(F_{\text{GetCurDcTickTime64}}\) instead.

The function returns the time of the current (last) tick in TwinCAT distributed clock system time format \((T\_DCTIME)\).

**FUNCTION** \(F_{\text{GetCurDcTickTime}}: T\_DCTIME\)

**VAR_INPUT**

\[\text{(*none*)}\]

**END_VAR**

**Example:**

```plaintext
PROGRAM MAIN
VAR
  tDC : T_DCTIME;
  sDC : STRING;
  tDCBack : T_DCTIME;
  sDCZero : STRING; (* DCTIME = zero time starts on 01.01.2000 *)
  tDCBackFromZero : T_DCTIME;
  tDCFromString : T_DCTIME;
```
Distributed Clocks

sDCBackFromString : STRING;
END_VAR

tDC := F_GetCurDcTickTime();
sDC := DCTIME_TO_STRING( tDC );
tDCBack := STRING_TO_DCTIME( sDC );

sDCZero := DCTIME_TO_STRING( U_LARGE_INTEGER(0, 0) );
tDCBackFromZero := STRING_TO_DCTIME( sDCZero );

tDCFromString := STRING_TO_DCTIME( '2007-03-09-11:31:09.223456789' );
sDCBackFromString := DCTIME_TO_STRING( tDCFromString );

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.2.7  

**F_GetCurExtTime**

The function returns the external time in TwinCAT distributed clock system time format (`T_DCTIME`).

**FUNCTION F_GetCurExtTime: T_DCTIME**

VAR_INPUT

<table>
<thead>
<tr>
<th>DcToExtTimeOffset : T_LARGE_INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DcToTcTimeOffset : T_LARGE_INTEGER</td>
</tr>
</tbody>
</table>
| END_VAR

**DcToExtTimeOffset**: Time offset between the TwinCAT distributed clock system time and an external time

**DcToTcTimeOffset**: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.2.8  

**FB_EcExtSyncCalcTimeDiff**

The function block `FB_EcExtSyncCalcTimeDiff` calculates the difference between external and internal time, taking into account the time offsets.

**Outdated function block**

This function block is outdated. Use the function block `FB_EcExtSyncCalcTimeDiff64` instead.

**FUNCTION FB_EcExtSyncCalcTimeDiff**

VAR_INPUT

| DcToExtTimeOffset : T_LARGE_INTEGER |
| DcToTcTimeOffset : T_LARGE_INTEGER  |
| ExtTime : T_DCTIME                 |
| IntTime : T_DCTIME                 |

**DcToExtTimeOffset**: Time offset between the TwinCAT distributed clock system time and an external time

**DcToTcTimeOffset**: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

**ExtTime**: External time

**IntTime**: Internal time
Distributed Clocks

VAR_IN_OUT

VAR_IN_OUT
DcToTcTimeOffset : T_LARGE_INTEGER;
DcToExtTimeOffset : T_LARGE_INTEGER;
ExtTime : T_DCTIME;
IntTime : T_DCTIME;
END_VAR

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time

ExtTime: External time in TwinCAT distributed clock system time format

IntTime: Internal time in TwinCAT distributed clock system time format

VAR_OUTPUT

VAR_OUTPUT
nTimeDiff : UDINT; (*with difference greater than 32 bit timeDiff = 0xffffffff*)
nOffsetFromSyncMaster : DINT; (*less than 32 bit int Offset = 0x80000000, greater than 32 bit int Offset = 0x7FFFFFFF*)
END_VAR

nTimeDiff: If the difference is less than 32 bit, the time difference is returned. If the difference is greater than 32 bit, 16#FFFFFFFF is returned.

nOffsetFromSyncMaster: If the difference is greater than 32 bit and the offset between internal and DC time is less than 32 bit, 16#80000000 is returned.
If the difference is greater than 32 bit and the offset between internal and DC time is greater than 32 bit, 16#7FFFFFFF is returned.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

10.4.2.9  FB_EcExtSyncCheck

Outdated function block

This function block is outdated. Use the function block FB_EcExtSyncCheck64 [84] instead.

The function block FB_EcExtSyncCheck checks whether the internal and external clocks are synchronous. See function block FB_EcExtSyncCalcTimeDiff [95].

VAR_INPUT

VAR_INPUT
nSyncWindow : UDINT;
bNotConnected : BOOL;
END_VAR

nSyncWindow: Time window within which the internal and external clock are regarded as synchronous.
bNotConnected: TRUE = connection to external clock is interrupted.
VAR_IN_OUT

VAR_IN_OUT
DcToTcTimeOffset : T_LARGE_INTEGER;
DcToExtTimeOffset : T_LARGE_INTEGER;
ExtTime : T_DCTIME;
IntTime : T_DCTIME;
END_VAR

DcToTcTimeOffset: Time offset between the TwinCAT distributed clock system time and the TwinCAT system time

DcToExtTimeOffset: Time offset between the TwinCAT distributed clock system time and an external time

ExtTime: External time in TwinCAT distributed clock system time format

IntTime: Internal time in TwinCAT distributed clock system time format

VAR_OUTPUT

VAR_OUTPUT
bSynchronized : BOOL;
nTimeDiff : UDINT;
nOffsetFromSyncMaster : DINT;
END_VAR

bSynchronized: TRUE = external and internal clock are synchronous

nTimeDiff: Current time difference between the two clocks

nOffsetFromSyncMaster: Offset to sync master

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2 EtherCAT</td>
</tr>
</tbody>
</table>

10.4.3 DCTIME64_TO_FILETIME

The function converts a 64-bit distributed clock system time variable of type T_DCTIME64 [108] to a 64-bit Windows file time variable of type T_FILETIME.

FUNCTION DCTIME64_TO_FILETIME: T_FILETIME

VAR_INPUT
in : T_DCTIME64;
END_VAR;

in: The distributed clock system time variable to be converted

Example:

PROGRAM P_TEST
VAR
ft : T_FILETIME;
dct : T_DCTIME64;
END_VAR

dct := F_GetCurDcTickTime64();
ft := DCTIME64_TO_FILETIME(dct);
**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

**10.4.4 FILETIME_TO_DCTIME64**

The function converts a 64-bit Windows file time variable of type T_FILETIME to a 64-bit distributed clock system time variable of type T_DCTIME64. In the event of a conversion error the function returns the value zero.

FUNCTION FILETIME_TO_DCTIME64: T_DCTIME64

VAR_INPUT

in : T_FILETIME;

END_VAR

in: The Windows file time variable to be converted.

Example:

PROGRAM P_TEST
VAR
  fbSysFileTime : GETSYSTEMTIME;
  ft : T_FILETIME;
  dct : T_DCTIME64;
END_VAR

fbSysFileTime(timeLoDW=>ft.dwLowDateTime, timeHiDW=>ft.dwHighDateTime);
dct := FILETIME_TO_DCTIME64(ft);

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
11 [Obsolete]

11.1 F_GetVersionTcEtherCAT

Outdated function

This function is outdated. Use the global structure instance stLibVersion_Tc2_EtherCAT instead.

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

FUNCTION F_GetVersionTcEtherCAT : UINT

VAR_INPUT
  nVersionElement : INT;
END_VAR

nVersionElement : Version element to be read. Possible parameters:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
12 Data types

12.1 E_EcAdressingType

Addressing in EtherCAT is either position-dependent (eAdressingType_AutoInc), based on a fixed, configured address (eAdressingType_Fixed) or applies to all slaves (eAdressingType_Broadcast).

```
TYPE E_EcAdressingType :
  { eAdressingType_AutoInc:=1, (* Adress slave by it's position. (adp = 1-
    position, 1.Slave = 0, 2.Slave = 0xffff(-1) etc) *)
    (* EtherCAT commands: APRD, APWR, APRW *)
  eAdressingType_Fixed, (* Adress slave by configured ethercat slave address (adp = configured address
    ) *)
    (* EtherCAT commands: FPRD, FPWR, FPRW *)
  eAdressingType_Broadcast (* Adress all slaves. *)
    (* EtherCAT commands: BRD, BWR, BRW *)
  );
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.2 E_EcFoeMode

Access mode for the “File access over EtherCAT” mailbox protocol.

```
TYPE E_EcFoeMode :
  { eFoeMode_Write := 1,
    eFoeMode_Read
  );
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.3 E_EcMbxProtType

Supported EtherCAT mailbox protocol types.

```
TYPE E_EcMbxProtType:
  { eEcMbxProt_CoE := 3, (* CANopen over EtherCAT *)
    eEcMbxProt_FoE := 4, (* File over EtherCAT *)
    eEcMbxProt_SoE := 5 (* Servo Drive Profile over EtherCAT *)
  );
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
12.4 ST_EcCrcError

Structure containing the CRC error counters of the individual ports (A, B and C) of an EtherCAT slave device.

```
TYPE ST_EcCrcError :
  STRUCT
    portA : UDINT;
    portB : UDINT;
    portC : UDINT;
  END_STRUCT
END_TYPE
```

- **portA**: CRC error counter of Port A
- **portB**: CRC error counter of Port B
- **portC**: CRC error counter of Port C

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.5 ST_EcCrcErrorEx

Structure containing the CRC error counters of the individual ports (A, B, C and D) of an EtherCAT slave device.

```
TYPE ST_EcCrcErrorEx :
  STRUCT
    portA : UDINT;
    portB : UDINT;
    portC : UDINT;
    portD : UDINT;
  END_STRUCT
END_TYPE
```

- **portA**: CRC error counter of Port A
- **portB**: CRC error counter of Port B
- **portC**: CRC error counter of Port C
- **portD**: CRC error counter of Port D

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.6 ST_EcLastProtErrInfo

The structure ST_EcLastProtErrInfo contains additional error information relating to the most recent EtherCAT mailbox protocol error.

```
TYPE ST_EcSlaveState:
  STRUCT
    ownAddr : ST_AmsAddr;
    orgAddr : ST_AmsAddr;
    errCode : UDINT;
    binDesc : ARRAY[0..MAX_STRING_LENGTH] OF BYTE;
  END_STRUCT
END_TYPE
```

- **ownAddr**: Own AMS address (address of the communication device that queries the error information).
Data types

**orgAddr:** AMS address of the error originator (address of communication device that has triggered or caused the protocol error).

**errCode:** Mailbox protocol error number (SoE, CoE, FoE error code).

**binDesc:** Additional error information as binary data. The additional error information is device-specific and can include a string or binary data, for example.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.7 **ST_EcMasterStatistic**

```plaintext
TYPE ST_EcMasterStatistic :
STRUCT
  nSysTime : UDINT;
  nCycFrameCnt : UDINT;
  nCycFrameMissedCnt : UDINT;
  nQueuedFrameCnt : UDINT;
  nQueuedFrameMissedCnt : UDINT;
END_STRUCT
END_TYPE
```

- **nSysTime:** System time in µs
- **nCycFrameCnt:** Number of cyclic EtherCAT frames
- **nCycFrameMissedCnt:** Number of lost cyclic EtherCAT frames
- **nQueuedFrameCnt:** Number of acyclic EtherCAT frames
- **nQueuedFrameMissedCnt:** Number of lost acyclic EtherCAT frames

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.8 **ST_EcSlaveConfigData**

The structure **ST_EcSlaveConfigData** contains the EtherCAT configuration data for an EtherCAT slave device.

```plaintext
TYPE ST_EcSlaveConfigData:
STRUCT
  nEntries : WORD;
  nAddr : WORD;
  sType : STRING[15];
  sName : STRING[31];
  nDevType : DWORD;
  stSlaveIdentity : ST_EcSlaveIdentity;
  nMailboxOutSize : WORD;
  nMailboxInSize : WORD;
  nLinkStatus : BYTE;
END_STRUCT
END_TYPE
```

- **nEntries:** used internally
- **nAddr:** Address of an EtherCAT slave
- **sType:** EtherCAT type of a slave
- **sName:** Name of an EtherCAT slave
nDevType: EtherCAT device type of a slave

stSlaveIdentity: Identity of an EtherCAT slave (see ST_EcSlaveIdentity [103])

nMailboxOutSize: Mailbox OutSize of an EtherCAT slave.

nMailboxInSize: Mailbox InSize of an EtherCAT slave.

nLinkStatus: Link status of an EtherCAT slave.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.9 ST_EcSlaveIdentity

The structure ST_EcSlaveIdentity contains the EtherCAT identity data of an EtherCAT slave device.

```
TYPE ST_EcSlaveIdentity :
STRUCT
  vendorId    : UDINT;
  productCode : UDINT;
  revisionNo  : UDINT;
  serialNo    : UDINT;
END_STRUCT
END_TYPE
```

vendorId: Vendor-ID of the slave device.

productCode: Product code of the slave device.

revisionNo: Indicates the revision number of the slave device.

serialNo: Indicates the serial number of the slave device.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.10 ST_EcSlaveScannedData

The structure ST_EcSlaveScannedData contains the EtherCAT configuration data of a scanned EtherCAT slave device.

```
TYPE ST_EcSlaveConfigData:
STRUCT
  nEntries     : WORD;
  nAddr        : WORD;
  stSlaveIdentity : ST_EcSlaveIdentity;
  nIdStatusReg : WORD;
END_STRUCT
END_TYPE
```

nEntries: used internally

nAddr: Address of an EtherCAT slave

stSlaveIdentity: Identity of an EtherCAT slave (see ST_EcSlaveIdentity [103])
**ndlStatusReg:** Link status of an EtherCAT slave from ESC register 0110/0111\textsubscript{Hex}, or 272/273\textsubscript{Dec}. Status 0 is displayed if the slave cannot be reached or is offline. The port number <=> socket/EBus contact assignment can be found in the respective device documentation. Unless described otherwise, port 0 is the left-hand EBus contact of an EL/ES terminal or the RJ45 socket of an EP box, port 1 is the right-hand outgoing EBus contact/RJ45 socket.

The bit meanings are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>internal use</td>
</tr>
<tr>
<td>2</td>
<td>internal use</td>
</tr>
<tr>
<td>3</td>
<td>internal use</td>
</tr>
<tr>
<td>4</td>
<td>physical link on Port 0 0: no link, 1: Link detected</td>
</tr>
<tr>
<td>5</td>
<td>physical link on Port 1 0: no link, 1: Link detected</td>
</tr>
<tr>
<td>6</td>
<td>physical link on Port 2 0: no link, 1: Link detected</td>
</tr>
<tr>
<td>7</td>
<td>physical link on Port 3 0: no link, 1: Link detected</td>
</tr>
<tr>
<td>8</td>
<td>Loop Port 0 0: Open, 1: Closed</td>
</tr>
<tr>
<td>9</td>
<td>Communication on Port 0 0: no stable communication, 1: Communication established</td>
</tr>
<tr>
<td>10</td>
<td>Loop Port 1 0: Open, 1: Closed</td>
</tr>
<tr>
<td>11</td>
<td>Communication on Port 1 0: no stable communication, 1: Communication established</td>
</tr>
<tr>
<td>12</td>
<td>Loop Port 2 0: Open, 1: Closed</td>
</tr>
<tr>
<td>13</td>
<td>Communication on Port 2 0: no stable communication, 1: Communication established</td>
</tr>
<tr>
<td>14</td>
<td>Loop Port 3 0: Open, 1: Closed</td>
</tr>
<tr>
<td>15</td>
<td>Communication on Port 3 0: no stable communication, 1: Communication established</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 v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 12.11 ST_EcSlaveState

The structure **ST_EcSlaveState** contains the EtherCAT status and the link status of an EtherCAT slave device.

```plaintext
TYPE ST_EcSlaveState:
  STRUCT
    deviceState :BYTE;
    linkState   :BYTE;
  END_STRUCT
END_TYPE
```

**deviceState:** EtherCAT status of a slave. The status can adopt one of the following values:
### Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_DEVICE_STATE_INIT</td>
<td>0x01</td>
<td>Init state</td>
</tr>
<tr>
<td>ECDEVICE_STATE_PREOP</td>
<td>0x02</td>
<td>Pre-operational state</td>
</tr>
<tr>
<td>ECDEVICE_STATE_BOOTSTRA</td>
<td>P</td>
<td>Bootstrap state</td>
</tr>
<tr>
<td>ECDEVICE_STATE_SAFEOP</td>
<td>0x04</td>
<td>Safe-operational state</td>
</tr>
<tr>
<td>ECDEVICE_STATE_OP</td>
<td>0x08</td>
<td>Operational state</td>
</tr>
</tbody>
</table>

In addition, the following bits can be set:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDEVICE_STATE_ERROR</td>
<td>0x10</td>
<td>State machine error in the EtherCAT slave</td>
</tr>
<tr>
<td>ECDEVICE_STATE_INVALID_V</td>
<td>PRS</td>
<td>Invalid vendor ID, product code, revision number or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>serial number</td>
</tr>
<tr>
<td>ECDEVICE_STATE_INITCMD_E</td>
<td>RROR</td>
<td>Error during sending of initialization commands.</td>
</tr>
<tr>
<td>ECDEVICE_STATE_DISABLED</td>
<td>0x80</td>
<td>Slave is disabled</td>
</tr>
</tbody>
</table>

### Link State

The link status of an EtherCAT slave can consist of an ORing of the following bits.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_LINK_STATE_OK</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>EC_LINK_STATE_NOT_PRESENT</td>
<td>0x01</td>
<td>No EtherCAT communication with the EtherCAT slave</td>
</tr>
<tr>
<td>EC_LINK_STATE_LINK_WITHOUT_COMM</td>
<td>0x02</td>
<td>Error at port X (specified through EC_LINK_STATE_PORT_A/B/C/D). The port has</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a link, but no communication is possible via this port.</td>
</tr>
<tr>
<td>EC_LINK_STATE_MISSING_LINK</td>
<td>0x04</td>
<td>Missing link at port X (specified through EC_LINK_STATE_PORT_A/B/C/D).</td>
</tr>
<tr>
<td>EC_LINK_STATE_ADDITIONAL_LINK</td>
<td>0x08</td>
<td>Additional link at port X (specified through EC_LINK_STATE_PORT_A/B/C/D).</td>
</tr>
</tbody>
</table>

### Requirements

#### Development Environment

<table>
<thead>
<tr>
<th>Target Platform</th>
<th>PLC Libraries to Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### Type

#### ST_EcSlaveStateBits

The type ST_EcSlaveStateBits contains the EtherCAT status and the link status of an EtherCAT slave device.

```plaintext
TYPE ST_EcSlaveStateBits:
STRUCT
  bInit    : BOOL;
  bPreop   : BOOL;
  bBootStrap : BOOL;
ENDSTRUCT
```

12.12 ST_EcSlaveStateBits

The structure ST_EcSlaveStateBits contains the EtherCAT status and the link status of an EtherCAT slave device.
Data types

```plaintext
bSafeOp   : BOOL;
bOp       : BOOL;
bError    : BOOL;
bInvVPRS  : BOOL;
bInitCmdError : BOOL;
bLinkNotPresent : BOOL;
bLinkWithoutComm : BOOL;
bLinkMissing : BOOL;
bAdditionalLink : BOOL;
bPortA   : BOOL;
bPortB   : BOOL;
bPortC   : BOOL;
bPortD   : BOOL;
END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.13  **ST_EcSlaveStateBitsEx**

The structure `ST_EcSlaveStateBitsEx` contains the EtherCAT status and the link status of an EtherCAT slave device.

```plaintext
TYPE ST_EcSlaveStateBitsEx:
STRUCT
  bInit      : BOOL;
bPreop     : BOOL;
bBootStrap : BOOL;
bSafeOp    : BOOL;
bOp        : BOOL;
bError     : BOOL;
bInvVPRS   : BOOL;
bInitCmdError : BOOL;
bDisabled  : BOOL;
bLinkNotPresent : BOOL;
bLinkWithoutComm : BOOL;
bLinkMissing : BOOL;
bAdditionalLink : BOOL;
bPortA : BOOL;
bPortB : BOOL;
bPortC : BOOL;
bPortD : BOOL;
END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.14  **ST_PortAddr**

The structure `ST_PortAddr` contains EtherCAT topology information for EtherCAT slave device. EtherCAT slave devices typically have 2 to 4 ports.

```plaintext
TYPE ST_PortAddr:
STRUCT
  portA : UINT;
  portB : UINT;
  portC : UINT;
  portD : UINT;
END_STRUCT
END_TYPE
```

`portA`: Address of the previous EtherCAT slave at port A of the current EtherCAT slave.
**portB**: Address of the optional subsequent EtherCAT slave at port B of the current EtherCAT slave

**portC**: Address of the optional subsequent EtherCAT slave at port C of the current EtherCAT slave

**portD**: Address of the optional subsequent EtherCAT slave at port D of the current EtherCAT slave

---

### 12.15 ST_TopologyDataEx

The structure **ST_TopologyDataEx** contains information on EtherCAT topology and hot-connect groups.

```plaintext
TYPE ST_TopologyDataEx:
STRUCT
  nOwnPhysicalAddr : UINT;
  nOwnAutoIncAddr  : UINT;
  stPhysicalAddr   : ST_PortAddr;
  stAutoIncAddr    : ST_PortAddr;
  aReserved1       : ARRAY [0..3] OF UDINT;
  nStatusBits      : DWORD;
  nHCSlaveCountCfg : UINT; (*nStatusBits.0 = TRUE: DCsupprt;.1 = TRUE: DC64supprt; .2=TRUE: Slave
                          Present following hot connect info requires runtime >= TC 2.11 R3 B2246 nStatusBits.3 = TRUE: HotCon
                          nectGroupStart; .4 = HotConnectSlave; .5 = TRUE: HotConnectInvalidB; .6 = TRUE: HotConnectInvalidC;
                          .7 = TRUE: HotConnectInvalidD*)
  nHCSlaveCountAct : UINT;
  aReserved2       : ARRAY [0..4] OF UDINT;
END_STRUCT
END_TYPE
```

- **nOwnPhysicalAddr**: Dedicated physical EtherCAT address of the EtherCAT slave device
- **nOwnAutoIncAddr**: Dedicated auto-increment EtherCAT address of the EtherCAT slave device
- **stPhysicalAddr**: Physical address information of the EtherCAT slave devices at port A…D
- **stAutoIncAddr**: Auto-increment address information of the EtherCAT slave devices at port A…D
- **aReserved1**: reserved
- **nStatusBits**:
  - nStatusBits.0 = TRUE: Distributed clocks are supported
  - nStatusBits.1 = TRUE: Distributed clocks are supported (64 bit)
  - nStatusBits.2 = TRUE: Slave is present
  - nStatusBits.3 = TRUE: Slave is start node of a Hot Connect group
  - nStatusBits.4 = TRUE: Slave is in a Hot Connect group
  - nStatusBits.5 = TRUE: Hot-connect is invalid at port B
  - nStatusBits.6 = TRUE: Hot-connect is invalid at port C
  - nStatusBits.7 = TRUE: Hot-connect is invalid at port D
- **nHCSlaveCountCfg**: Configured number of hot-connect group devices
- **nHCSlaveCountAct**: Found number of hot-connect group devices
- **aReserved2**: reserved

#### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

---

### 12.16 DCTIMESTRUCT

Structured TwinCAT distributed clock system time format. The smallest unit is a nanosecond. This data type represents the number of nanoseconds since 01.01.2000 (GMT).

```plaintext
TYPE DCTIMESTRUCT :
STRUCT
  wYear   : WORD;
  wMonth  : WORD;
END_STRUCT
```

---

TE1000  
Version: 1.5  
107
Data types

```plaintext
wDayOfWeek    : WORD;
wDay          : WORD;
wHour         : WORD;
wMinute       : WORD;
wSecond       : WORD;
wMilliseconds: WORD;
wMicroseconds : WORD;
wNanoseconds  : WORD;
END_STRUCT
END_TYPE

wYear         : Year: 2000 ~ 2584;
wMonth        : Month: 1 ~ 12 (January = 1, February = 2 etc.);
wDayOfWeek    : Day of the week: 0 ~ 6 (Sunday = 0, Monday = 1 etc.);
wDay          : The day of the month: 1 ~ 31;
wHour         : Hour: 0 ~ 23;
wMinute       : Minute: 0 ~ 59;
wSecond       : Second: 0 ~ 59;
wMilliseconds: Millisecond: 0 ~ 999;
wMicroseconds : Microsecond: 0 ~ 999;
wNanoseconds  : Nanosecond: 0 ~ 999;
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.17 T_DCTIME32

32-bit TwinCAT distributed clock system time format. The smallest unit is a nanosecond.

This 32-bit DC system time is formed from the full absolute 64-bit DC system time (T_DCTIME) by using only the lowest-order 32 bits. This means the property of an absolute unique time is lost, and it is assumed that this 32-bit time is only used within a narrow time window of ± 2,147 seconds around the current system time, to ensure that it is unambiguous. There are many applications in which this assumption is possible.

If this assumption is violated, errors may occur in the interpretation and further processing of this time.

```plaintext
TYPE T_DCTIME32 : UDINT;
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

12.18 T_DCTIME64

64-bit TwinCAT distributed clock system time format. The smallest unit is a nanosecond.

```plaintext
TYPE T_DCTIME64 : ULINT;
END_TYPE
```
Useful distributed clock system time constants | Description
---|---
EC_DCTIME_DELTA_OFFSET64 | Number of 100-nanosecond ticks between 01.01.1601 and 01.01.2000. This is the difference between the Windows file time and the distributed clock system time.

EC_DCTIME_DATEDELTA_OFFSET | Number of days that have passed between the year zero and 1 January 2000

EC_DCTIME_TICKSPERMSEC64 | Number of distributed clock system time nanoseconds per millisecond

EC_DCTIME_TICKSPERSEC64 | Number of distributed clock system time nanoseconds per second

EC_DCTIME_TICKSPERDAY64 | Number of distributed clock system time nanoseconds per day

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

### 12.19 T_DCTIME

Outdated data type

This data type is outdated. Use the data type T_DCTIME64 instead.

The data type T_DCTIME represents the distributed clock system time (abbreviated as DC time) as a linear 64-bit integer value. The time is expressed in nanoseconds since 1.1.2000 UTC.

The data type is represented as two 32-bit DWORD variables, so that it can easily be processed in the PLC. Operations (addition and subtraction of times) can be executed with ui64 functions from the Tc2_Utilities library.

```plaintext
TYPE T_DCTIME : T_ULARGE_INTEGER;
END_TYPE
```

Useful distributed clock system time constants | Description
---|---
EC_DCTIME_DELTA_OFFSET | Number of 100-nanosecond ticks between 01.01.1601 and 01.01.2000. This is the difference between the Windows file time and the distributed clock system time.

EC_DCTIME_DATEDELTA_OFFSET | Number of days that have passed between the year zero and 1 January 2000

EC_DCTIME_TICKSPERMSEC | Number of distributed clock system time nanoseconds per millisecond

EC_DCTIME_TICKSPERSEC | Number of distributed clock system time nanoseconds per second

EC_DCTIME_TICKSPERDAY | Number of distributed clock system time nanoseconds per day

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
12.20 T_HFoe

“File access over EtherCAT” handle. Before the handle can be used, it must be initialized once with the function block FB_EcFoeOpen [5.56]. The variables of this structured type must not be written directly.

```plaintext
TYPE T_HFoe :
  STRUCT
    sNetID : T_AmsNetId := '';
    nPort : T_AmsPort := 0;
    handle : UDINT := 0;
    eMode : E_EcFoeMode := eFoeMode_Write;
  END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
13 Constants

13.1 Global constants

VAR_GLOBAL CONSTANT

EC_AMSPORT_MASTER :UINT :=16#FFFF;
EC_MAX_SLAVES :UINT :=16#FFFF;

(*ethercat commands*)
EC_CMD_TYPE_APRD :BYTE :=1;
EC_CMD_TYPE_APRW :BYTE :=2;
EC_CMD_TYPE_APRR :BYTE :=3;
EC_CMD_TYPE_FPFR :BYTE :=4;
EC_CMD_TYPE_FPFW :BYTE :=5;
EC_CMD_TYPE_FPWR :BYTE :=6;
EC_CMD_TYPE_BRD :BYTE :=7;
EC_CMD_TYPE_BWR :BYTE :=8;
EC_CMD_TYPE_BRW :BYTE :=9;
EC_CMD_TYPE_LRD :BYTE :=10;
EC_CMD_TYPE_LRW :BYTE :=11;
EC_CMD_TYPE_LWR :BYTE :=12;

(* Device states *)
EC_DEVICE_STATE_MASK :BYTE :=16#0F;
EC_DEVICE_STATE_INIT :BYTE :=16#01;
EC_DEVICE_STATE_PRD0 :BYTE :=16#02;
EC_DEVICE_STATE_BOOTSTRAP :BYTE :=16#03;
EC_DEVICE_STATE_SAFEP :BYTE :=16#04;
EC_DEVICE_STATE_OP :BYTE :=16#08;
EC_DEVICE_STATE_ERROR :BYTE :=16#10;
EC_DEVICE_STATE_INVALID_VPRS :BYTE :=16#20;
EC_DEVICE_STATE_INITCMD_ERROR :BYTE :=16#40;

(* Link states *)
EC_LINK_STATE_OK :BYTE :=16#00;
EC_LINK_STATE_NOT_PRESENT :BYTE :=16#01;
EC_LINK_STATE_MISSING_LINK :BYTE :=16#02;
EC_LINK_STATE_ADDITIONAL_LINK :BYTE :=16#08;
EC_LINK_STATE_PORT_A :BYTE :=16#10;
EC_LINK_STATE_PORT_B :BYTE :=16#20;
EC_LINK_STATE_PORT_C :BYTE :=16#40;
EC_LINK_STATE_PORT_D :BYTE :=16#80;

(* Device/Link state IG/IO *)
EC_ADS_IGRP_MASTER_STATEMACHINE :UDINT :=16#00000003;
EC_ADS_IOFFS_MASTER_CURSTATE :UDINT :=16#00000100;
EC_ADS_IOFFS_MASTER_REQSTATE :UDINT :=16#00000101;
EC_ADS_IOFFS_MASTER_INTERNALSTATE :UDINT :=16#00000102;
EC_ADS_IGRP_MASTER_COUNT_SLAVE :UDINT :=16#00000006;
EC_ADS_IGRP_MASTER_COUNT_SLAVE :UDINT :=16#00000000;
EC_ADS_IGRP_MASTER_COUNT_PORT :UDINT :=16#00000001;
EC_ADS_IGRP_MASTER_COUNT_ROUTER :UDINT :=16#00000002;
EC_ADS_IGRP_MASTER_SLAVE_ADDRESSES :UDINT :=16#00000007;
EC_ADS_IGRP_MASTER_SENDCMD :UDINT :=16#00000008;
EC_ADS_IGRP_MASTER_STATEMACHINE :UDINT :=16#00000009;
EC_ADS_IGRP_MASTER_SLAVE.IDENTITY :UDINT :=16#00000011;
EC_ADS_IGRP_MASTER_SLAVE_CRC :UDINT :=16#00000012;
EC_ADS_IGRP_MASTER_SLAVE_ABIRAL_STATE_CHANGES :UDINT :=16#00000013;
EC_ADS_IGRP_MASTER_SLAVE_PRESENT :UDINT :=16#00000016;
EC_ADS_IGRP_MASTER_DEVICESTATE :UDINT :=16#00000045;
EC_ADS_IGRP_MASTER_COUNT_FRAME :UDINT :=16#00000048;

(* SoE IG/IO *)
EC_ADS_IGRP_ECAT_SOE :UDINT :=16#0000F420;
EC_ADS_IGRP_ECAT_SOE_LASTERROR :UDINT :=16#0000F421;

EC_SOE_ELEMENT_DATASTATE :BYTE :=16#01;
EC_SOE_ELEMENT_NAME :BYTE :=16#02;
EC_SOE_ELEMENT_ATTRIBUTE :BYTE :=16#04;
EC_SOE_ELEMENT_UNIT :BYTE :=16#08;
Constants

EC_SO_E_ELEMENT_MIN :BYTE :=16#10;
EC_SO_E_ELEMENT_MAX :BYTE :=16#20;
EC_SO_E_ELEMENT_VALUE :BYTE :=16#40;
EC_SO_E_ELEMENT_DEFAULT :BYTE :=16#80;

(* FoE IG/IO *)
EC_ADS_IGRP_FOE_FOPENREAD :UDINT :=16#0000F401;
EC_ADS_IGRP_FOE_FOPENWRITE :UDINT :=16#0000F402;
EC_ADS_IGRP_FOE_FCLOSE :UDINT :=16#0000F403;
EC_ADS_IGRP_FOE_FREAD :UDINT :=16#0000F404;
EC_ADS_IGRP_FOE_FWRITE :UDINT :=16#0000F405;
EC_ADS_IGRP_FOE_PROGRESSINFO :UDINT :=16#0000F406;
EC_ADS_IGRP_FOE_LASTERROR :UDINT :=16#0000F407;

(* CoE IG/IO *)
EC_ADS_IGRP_CANOPEN_SDO :UDINT :=16#0000F302;
EC_ADS_IGRP_CANOPEN_SDO_LASTERROR :UDINT :=16#0000F303;

EC_DCTIME_DATEDELTA_OFFSET : DWORD := 730120; (* Number of past days since year zero until 1 January 2000 *)
EC_DCTIME_DELTA_OFFSET : T_ULARGE_INTEGER := ( dwHighPart := 16#01BF53EB, dwLowPart := 16#256D4000 ) ; (* Number of 100ns ticks between 1.1.1601 and 1.1.2000 *)
EC_DCTIME_TICKS_PER_MILLISECOND : T_ULARGE_INTEGER := ( dwHighPart := 16#00000000, dwLowPart := 16#0000F240); (* Number of nanosecond ticks per millisecond *)
EC_DCTIME_TICKS_PER_SEC : T_ULARGE_INTEGER := ( dwHighPart := 16#00000000, dwLowPart := 16#3B9ACA00); (* Number of nanosecond ticks per second *)
EC_DCTIME_TICKS_PER_DAY : T_ULARGE_INTEGER := ( dwHighPart := 16#00004E94, dwLowPart := 16#914F0000); (* Number of nanosecond ticks per day *)

bSeqReadDrvAttrAndValue : BOOL := FALSE;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>

13.2 Library version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

Global_Version

VAR_GLOBAL CONSTANT
  stLibVersion_Tc2_EtherCAT : ST_LibVersion;
END_VAR

stLibVersion_Tc2_EtherCAT: Version information of the Tc2_EtherCAT library (type: ST_LibVersion)

To check whether the version you have is the version you need, use the function F_CmpLibVersion (defined in the Tc2_System library).

All other options for comparing library versions, which you may know from TwinCAT 2, are outdated!
13.3 EtherCAT mailbox protocol error codes

VAR_GLOBAL CONSTANT

(* FoE mailbox protocol error codes *)
EC_FOE_PROTERR_NOTDEFINED : UDINT := 0;
EC_FOE_PROTERR_NOTFOUND : UDINT := 1;
EC_FOE_PROTERR_ACCESS : UDINT := 2;
EC_FOE_PROTERR_DISKFULL : UDINT := 3;
EC_FOE_PROTERR_ILLEGAL : UDINT := 4;
EC_FOE_PROTERR_PACKENO : UDINT := 5;
EC_FOE_PROTERR_EXISTS : UDINT := 6;
EC_FOE_PROTERR_NOUSER : UDINT := 7;
EC_FOE_PROTERR_BOOTSTRAPONLY : UDINT := 8;
EC_FOE_PROTERR_NOTINBOOTSTRAP : UDINT := 9;
EC_FOE_PROTERR_INVALIDPASSWORD : UDINT := 10;

(* CoE mailbox protocol error codes *)
EC_COE_PROTERR_TOGGLE : UDINT := 16#05030000; (* Toggle bit not alternated. *)
EC_COE_PROTERR_TIMEOUT : UDINT := 16#05040000; (* SDO protocol timed out. *)
EC_COE_PROTERR_CCS_SCS : UDINT := 16#05040001; (* Client/server command specifier not valid or unknown. *)
EC_COE_PROTERR_BLK_SIZE : UDINT := 16#05040002; (* Invalid block size (block mode only). *)
EC_COE_PROTERR_SEQNO : UDINT := 16#05040003; (* Invalid sequence number (block mode only). *)
EC_COE_PROTERR_CRC : UDINT := 16#05040004; (* CRC error (block mode only). *)
EC_COE_PROTERR_MEMORY : UDINT := 16#05040005; (* Out of memory. *)
EC_COE_PROTERR_ACCESS : UDINT := 16#06010000; (* Unsupported access to an object. *)
EC_COE_PROTERR_WRITEONLY : UDINT := 16#06010001; (* Attempt to write a read only object. *)
EC_COE_PROTERR_INDEX : UDINT := 16#06020000; (* Object does not exist in the object dictionary. *)
EC_COE_PROTERR_PDO_MAP : UDINT := 16#06040041; (* Object cannot be mapped to the PDO. *)
EC_COE_PROTERR_PDO_LEN : UDINT := 16#06040042; (* The number and length of the objects to be mapped would exceed PDO length. *)
EC_COE_PROTERR_P_INCOMP : UDINT := 16#06040043; (* General parameter incompatibility reason. *)
EC_COE_PROTERR_I_INCOMP : UDINT := 16#06040044; (* General internal incompatibility in the device. *)
EC_COE_PROTERR_HARDWARE : UDINT := 16#06060000; (* Access failed due to an hardware error. *)
EC_COE_PROTERR_DATA_SIZE : UDINT := 16#06070010; (* Data type does not match, length of service parameter does not match *)
EC_COE_PROTERR_DATA_SIZE1 : UDINT := 16#06070012; (* Data type does not match, length of service parameter too high *)
EC_COE_PROTERR_DATA_SIZE2 : UDINT := 16#06070013; (* Data type does not match, length of service parameter too low *)
EC_COE_PROTERR_OFFSET : UDINT := 16#06090011; (* Sub-index does not exist. *)
EC_COE_PROTERR_DATA_RANGE : UDINT := 16#06090030; (* Value range of parameter exceeded (only for write access). *)
EC_COE_PROTERR_DATA_RANGE1 : UDINT := 16#06090031; (* Value of parameter written too high. *)
EC_COE_PROTERR_DATA_RANGE2 : UDINT := 16#06090032; (* Value of parameter written too low. *)
EC_COE_PROTERR_MINMAX : UDINT := 16#06090036; (* Maximum value is less than minimum value. *)
EC_COE_PROTERR_GENERATE : UDINT := 16#08000000; (* General error *)
EC_COE_PROTERR_TRANSFER : UDINT := 16#08000020; (* Data cannot be transferred or stored to the application. *)
EC_COE_PROTERR_TRANSFER1 : UDINT := 16#08000021; (* Data cannot be transferred or stored to the application because of local control. *)
EC_COE_PROTERR_TRANSFER2 : UDINT := 16#08000022; (* Data cannot be transferred or stored to the application because of the present device state. *)
EC_COE_PROTERR_DICTIONARY : UDINT := 16#08000023; (* Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error). *)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_EtherCAT</td>
</tr>
</tbody>
</table>
More Information:
[www.beckhoff.com/te1000]