acontis technologies GmbH

SOFTWARE

EC-Simulator

User Manual

V3.3

AT3503 **Edition: July 22, 2026**

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

© Copyright **acontis technologies GmbH**

Neither this document nor excerpts therefrom may be reproduced, transmitted, or conveyed to third parties by any means whatever without the express permission of the publisher. At the time of publication, the functions described in this document and those implemented in the corresponding hardware and/or software were carefully verified; nonetheless, for technical reasons, it cannot be guaranteed that no discrepancies exist. This document will be regularly examined so that corrections can be made in subsequent editions. Note: Although a product may include undocumented features, such features are not considered to be part of the product, and their functionality is therefore not subject to any form of support or guarantee.

Contents

1 Introduction **9**
- 1.1 What is EtherCAT®? . . . 9
- 1.2 EC-Simulator - Features . . . 9
- 1.3 Editions (HiL / SiL) . . . 11
  - 1.3.1 Hardware-in-Loop (HiL) Simulation . . . 11
  - 1.3.2 Software-in-the-Loop (SiL) Simulation . . . 11
- 1.4 Protected version . . . 12
  - 1.4.1 Licensing procedure for Development Licenses . . . 12
  - 1.4.2 Licensing procedure for Runtime Licenses . . . 13
- 1.5 Known restrictions . . . 13
- 1.6 License . . . 14
  - 1.6.1 EC-Simulator license . . . 14
  - 1.6.2 Free Open Source Software contained in EC-Simulator . . . 14
  - 1.6.3 Free Open Source Software supported by EC-Simulator . . . 15
- 1.7 Versioning . . . 15

2 Getting Started **16**
- 2.1 EC-Simulator Architecture . . . 16
- 2.2 Extended EtherCAT® Network Configuration (EXI) . . . 16
- 2.3 SiL Simulation Software architecture . . . 17
- 2.4 Mixed mode of real SubDevices and simulated SubDevices . . . 17
- 2.5 Operating system configuration . . . 18
- 2.6 Using emllSimulator (SiL) . . . 19
  - 2.6.1 Simulator Ethernet Driver parameters . . . 19
- 2.7 EC-Simulator Software Development Kit (SDK) . . . 21
  - 2.7.1 OS Compiler settings . . . 21

3 Software Integration **23**
- 3.1 EcSimulatorSilDemo . . . 23
  - 3.1.1 File reference . . . 23
  - 3.1.2 Logging . . . 23
- 3.2 Process data update and synchronization . . . 24
  - 3.2.1 Forced Data, Firmware Data, DPRAM . . . 25
  - 3.2.2 EC-Simulator stack as process data memory provider . . . 25
- 3.3 Accessing process data in the application . . . 27
  - 3.3.1 Process Data Access Functions selection . . . 27
  - 3.3.2 Process variables' offset and size . . . 28
- 3.4 Hot Connect . . . 30
  - 3.4.1 Configured Station Alias . . . 30
- 3.5 Cable Redundancy . . . 31
- 3.6 Running EcSimulatorSilDemo with Cable Redundancy . . . 31
  - 3.6.1 Linux (EcSimulatorSilDemo) . . . 31
  - 3.6.2 EcMasterDemoPython (SiL) . . . 32
- 3.7 Error detection and diagnosis . . . 32
- 3.8 RAS-Server for EC-Lyser and EC-Engineer . . . 32
  - 3.8.1 Integration Requirements . . . 32
  - 3.8.2 Pseudo Code . . . 33
  - 3.8.3 Required API Calls . . . 34
- 3.9 Motion . . . 42
  - 3.9.1 Internal DS402 Simulation . . . 42
  - 3.9.2 EcSimulatorSilDemoMotion . . . 42
  - 3.9.3 Command line parameters . . . 43

4 Platform and Operating Systems (OS) **44**

4.1 tenAsys INtime . . . 44
4.1.1 OS Compiler settings . . . 44
4.2 Linux . . . 45
4.2.1 OS optimizations . . . 45
4.2.2 atemsys kernel module . . . 46
4.2.3 Unbind Ethernet Driver instance . . . 47
4.2.4 Docker . . . 48
4.2.5 OS Compiler settings . . . 49
4.2.6 Build using cmake on Linux . . . 50
4.2.7 Cross-platform development under Windows . . . 50
4.3 QNX Neutrino . . . 52
4.3.1 Thread priority . . . 52
4.3.2 Unbind Ethernet Driver instance . . . 52
4.3.3 IOMMU/SMMU support . . . 52
4.3.4 OS Compiler settings . . . 52
4.4 IntervalZero RTX . . . 54
4.4.1 OS Compiler settings . . . 54
4.5 Windriver VxWorks . . . 55
4.5.1 VxWorks native . . . 55
4.5.2 SNARF Ethernet Driver . . . 56
4.5.3 OS Compiler settings . . . 56
4.6 Microsoft Windows . . . 57
4.6.1 OS Compiler settings . . . 57
4.6.2 RtaccDevice for Real-time Ethernet Driver . . . 57
4.7 Xenomai . . . 70
4.7.1 OS compiler settings . . . 70

5 Real-time Ethernet Driver 72
5.1 Real-time Ethernet Driver initialization . . . 72
5.1.1 Real-time Ethernet Driver instance selection via PCI location . . . 76
5.2 Intel Pro/1000 - emllIntelGbe . . . 76
5.2.1 TTS Feature . . . 77
5.2.2 Supported PCI devices . . . 78
5.3 Intel Pro/100 - emllI8255x . . . 79
5.3.1 Supported PCI devices . . . 79
5.4 Broadcom Genet - emllBcmGenet . . . 80
5.5 Broadcom NetXtreme - emllBcmNetXtreme . . . 80
5.5.1 Supported PCI devices . . . 81
5.6 Beckhoff CCAT - emllCCAT . . . 81
5.6.1 Supported PCI devices . . . 82
5.7 Texas Instruments CPSW - emllCPSW . . . 82
5.7.1 CPSW usage under Linux . . . 83
5.8 Linux DPDK - emllDpdk . . . 84
5.8.1 Linux System Requirements . . . 84
5.8.2 Huge pages setup . . . 85
5.8.3 DPDK for PCI Network Adapter . . . 85
5.8.4 DPDK for DPAA . . . 85
5.8.5 DPDK for ENETC4 . . . 88
5.8.6 Limitations . . . 89
5.9 DW3504 - emllDW3504 . . . 89
5.9.1 Supported PCI devices . . . 91
5.10 Freescale TSEC / eTSEC - emllETSEC . . . 92
5.10.1 ETSEC supported MAC's . . . 93
5.10.2 Shared MII bus . . . 93
5.10.3 Locking . . . 94
5.10.4 Link check . . . 94
5.10.5 Fixed Link . . . 94
5.11 Freescale FslFec - emllFslFec . . . 95

5.12 Cadence GEM/MACB - emllGEM . . . 96
5.13 INtime HPE - emllHPE . . . 98
5.14 Texas Instruments ICSS - emllICSS . . . 98
5.14.1 TTS Feature . . . 100
5.14.2 TI AM335x ICEV2 . . . 100
5.14.3 TI AM57xx IDK . . . 100
5.14.4 AM5728 IDK and AM5718 IDK boards and Technical Limitations . . . 100
5.15 Windows NDIS - emllNdis . . . 101
5.16 Windows WinPcap - emllPcap . . . 102
5.16.1 WinPcap, Npcap support . . . 102
5.17 RDC R6040 - emllR6040 . . . 103
5.17.1 Supported PCI devices . . . 104
5.18 emllRemote . . . 104
5.19 Realtek RTL8169 - emllRTL8169 . . . 105
5.19.1 RTL8169 usage under Linux . . . 105
5.19.2 Supported PCI devices . . . 105
5.20 VxWorks SNARF - emllSNARF . . . 106
5.21 Linux SockRaw - emllSockRaw . . . 106
5.22 Linux SockXdp - emllSockXdp . . . 107
5.22.1 Linux System Requirements . . . 108
5.22.2 Getting started . . . 109
5.23 Windows TAP - emllTap . . . 109

6 Application programming interface, reference 110
6.1 Header files . . . 110
6.2 Generic API return status values . . . 110
6.3 Multiple EtherCAT® Network Support . . . 110
6.3.1 Overview . . . 110
6.3.2 Licensing . . . 111
6.4 General functions . . . 111
6.4.1 esInitSimulator . . . 111
6.4.2 esDeinitSimulator . . . 115
6.4.3 esGetSimulatorParms . . . 116
6.4.4 esSetSimulatorParms . . . 116
6.4.5 esSetLogParms . . . 116
6.4.6 esGetSrcMacAddress . . . 117
6.4.7 esSetLicenseKey (HiL) . . . 117
6.4.8 esSetOemKey (HiL) . . . 118
6.4.9 esConfigureNetwork . . . 118
6.4.10 esGetCfgSlaveInfo . . . 120
6.4.11 esGetCfgSlaveSmInfo . . . 124
6.4.12 esSetSlaveSscApplication . . . 125
6.4.13 esRegisterClient . . . 128
6.4.14 esUnregisterClient . . . 129
6.4.15 esExecJob . . . 129
6.4.16 esGetMasterState . . . 133
6.4.17 esIoCtl . . . 133
6.4.18 esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE . . . 134
6.4.19 esIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB . . . 134
6.4.20 esIoCtl - EC_IOCTL_ISLINK_CONNECTED . . . 135
6.4.21 esIoCtl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO . . . 136
6.4.22 esIoCtl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES . . . 137
6.4.23 esIoCtl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL . . . 137
6.4.24 esIoCtl - EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL . . . 138
6.4.25 esIoCtl - EC_IOCTL_GET_LINKLAYER_MODE . . . 138
6.4.26 esIoCtl - EC_LINKIOCTL_XXXX . . . 139
6.4.27 esIoCtl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS . . . 139
6.4.28 esIoCtl - EC_LINKIOCTL_GET_SPEED . . . 139

6.4.29 esIoCtl - EC_LINKIOCTL_GET_PCI_INFO . . . 140
6.4.30 esIoCtl - EC_LINKIOCTL_FORCE_LINK_STATUS . . . 142
6.4.31 esGetVersion . . . 143
6.4.32 esGetText . . . 143
6.4.33 esGetMemoryUsage . . . 143
6.4.34 esLogFrameEnable . . . 144
6.4.35 esLogFrameDisable . . . 145
6.5 Process Data Access Functions . . . 145
6.5.1 esGetProcessImageInputPtr . . . 145
6.5.2 esGetProcessImageOutputPtr . . . 145
6.5.3 EC_COPYBITS . . . 146
6.5.4 EC_GET_FRM_WORD . . . 147
6.5.5 EC_GET_FRM_DWORD . . . 147
6.5.6 EC_GET_FRM_QWORD . . . 148
6.5.7 EC_SET_FRM_WORD . . . 148
6.5.8 EC_SET_FRM_DWORD . . . 148
6.5.9 EC_SET_FRM_QWORD . . . 149
6.5.10 EC_GETBITS . . . 149
6.5.11 EC_SETBITS . . . 149
6.5.12 EC_COPYBIT . . . 150
6.5.13 EC_TESTBIT . . . 150
6.5.14 EC_SETBIT . . . 150
6.5.15 EC_CLRBIT . . . 151
6.5.16 esGetProcessData . . . 151
6.5.17 esSetProcessData . . . 152
6.5.18 esSetProcessDataBits . . . 152
6.5.19 esGetProcessDataBits . . . 153
6.6 Notifications . . . 153
6.6.1 Notification handler . . . 153
6.6.2 EC_NOTIFY_SLAVE_PRESENCE . . . 154
6.6.3 EC_NOTIFY_SLAVE_STATECHANGED . . . 155
6.6.4 EC_NOTIFY_STATECHANGED . . . 155
6.6.5 EC_NOTIFY_HC_TOPOCHGDONE . . . 156
6.6.6 EC_NOTIFY_SLAVE_ERROR_STATUS_INFO . . . 156
6.6.7 EC_NOTIFY_EEPROM_OPERATION . . . 157
6.6.8 EC_NOTIFY_FRAME_RESPONSE_ERROR . . . 159
6.6.9 esNotifyApp . . . 160
6.7 Network operation functions . . . 161
6.7.1 esConnectPorts . . . 161
6.7.2 esDisconnectPort . . . 161
6.7.3 esPowerSlave . . . 162
6.8 Error simulation functions . . . 162
6.8.1 esSetErrorAtSlavePort . . . 162
6.8.2 esSetErrorGenerationAtSlavePort . . . 163
6.8.3 esResetErrorGenerationAtSlavePorts . . . 164
6.8.4 esSetLinkDownAtSlavePort . . . 164
6.8.5 esSetLinkDownGenerationAtSlavePort . . . 164
6.8.6 esResetLinkDownGenerationAtSlavePorts . . . 165
6.8.7 esLogFrameEnableAtSlavePort . . . 165
6.8.8 esLogFrameDisableAtSlavePort . . . 166
6.8.9 esSendSlaveCoeEmergency . . . 166
6.8.10 esSetSimSlaveState . . . 167
6.9 SubDevice control and status functions . . . 169
6.9.1 esGetNumConfiguredSlaves . . . 169
6.9.2 esGetNumConnectedSlaves . . . 169
6.9.3 esGetSlaveId . . . 170
6.9.4 esGetSlaveIdAtPosition . . . 170
6.9.5 esGetSlaveState . . . 171

6.9.6 esSetSlaveState . . . 171
6.9.7 esIsSlavePresent . . . 172
6.9.8 esGetSlaveProp . . . 173
6.9.9 esGetSlavePortState . . . 173
6.9.10 esGetProcessVarInfoNumOf, esGetProcessVarInfoEx . . . 174
6.9.11 esGetSlaveInpVarInfoNumOf . . . 176
6.9.12 esGetSlaveOutpVarInfoNumOf . . . 176
6.9.13 esGetSlaveInpVarInfo . . . 177
6.9.14 esGetSlaveInpVarInfoEx . . . 178
6.9.15 esGetSlaveOutpVarInfo . . . 179
6.9.16 esGetSlaveOutpVarInfoEx . . . 180
6.9.17 esGetSlaveInpVarByObjectEx . . . 180
6.9.18 esGetSlaveOutpVarByObjectEx . . . 181
6.9.19 esFindInpVarByName . . . 182
6.9.20 esFindInpVarByNameEx . . . 182
6.9.21 esFindOutpVarByName . . . 182
6.9.22 esFindOutpVarByNameEx . . . 183
6.9.23 esWriteSlaveRegister . . . 183
6.9.24 esReadSlaveRegister . . . 184
6.9.25 esReadSlaveEEPRom . . . 185
6.9.26 esWriteSlaveEEPRom . . . 186
6.9.27 esGetSimSlaveInfo . . . 186
6.9.28 esGetBusSlaveInfo . . . 188
6.9.29 esReadSlaveIdentification . . . 191
6.9.30 esGetSlaveStatistics . . . 192
6.9.31 esClearSlaveStatistics . . . 193
6.10 ADS over EtherCAT® (AoE) . . . 193
6.10.1 esAoeGetSlaveNetId . . . 193
6.10.2 esAoeRead . . . 194
6.10.3 esAoeWrite . . . 195
6.10.4 esAoeReadWrite . . . 196
6.10.5 esSetSlaveAoeObjectTransferCallbacks . . . 197
6.10.6 AoE Simulator and MainDevice Example . . . 197
6.11 CAN application protocol over EtherCAT® (CoE) . . . 202
6.11.1 esExtendSlaveCoeObjectDictionary . . . 202
6.11.2 esDeleteSlaveCoeObject . . . 203
6.11.3 esClearSlaveCoeObjectDictionary . . . 203
6.11.4 esResetSlaveCoeObjectDictionary . . . 204
6.11.5 esSetSlaveCoeObjectTransferCallbacks . . . 204
6.11.6 esCoeSdoDownload . . . 207
6.11.7 esCoeSdoUpload . . . 208
6.11.8 esCoeGetODList . . . 209
6.11.9 esCoeGetODListReq . . . 210
6.11.10 esCoeGetObjectDesc . . . 211
6.11.11 esCoeGetObjectDescReq . . . 212
6.11.12 esCoeGetEntryDesc . . . 213
6.11.13 esCoeGetEntryDescReq . . . 215
6.11.14 CoE transfer Simulator and MainDevice Example . . . 216
6.12 Ethernet over EtherCAT® (EoE) . . . 219
6.12.1 esEoeSendFrame . . . 219
6.12.2 esGetCfgSlaveEoeInfo . . . 220
6.12.3 EoE Ping Example . . . 223
6.13 File access over EtherCAT® (FoE) . . . 227
6.14 Vendor specific access over EtherCAT® (VoE) . . . 231
6.14.1 esVoeSend . . . 231
6.14.2 esSetVoeReceiveCallback . . . 231
6.14.3 VoE Receive Example . . . 232
6.15 Distributed Clocks (DC) . . . 234

7 Error Codes **235**
7.1 Groups . . . 235
7.2 Generic Error Codes . . . 235
7.3 DCM Error Codes . . . 246
7.4 ADS over EtherCAT® (AoE) Error Codes . . . 248
7.5 CAN application protocol over EtherCAT® (CoE) SDO Error Codes . . . 250
7.6 File Transfer over EtherCAT® (FoE) Error Codes . . . 253
7.7 Servo Drive Profil over EtherCAT® (SoE) Error Codes . . . 255
7.8 Remote API Error Codes . . . 259

1 Introduction

1.1 What is EtherCAT®?

EtherCAT® (Ethernet for Control Automation Technology) is a high-performance Ethernet Fieldbus technology that provides a reliable, efficient, and cost-effective communication solution for a wide variety of industrial automation applications. Originally developed as an open technology by Beckhoff Automation in 2003, and subsequently turned over to an independent organization known as the EtherCAT® Technology Group, EtherCAT® has since become one of the most widely used industrial Ethernet protocols in the world.

See also:

A comprehensive introduction to EtherCAT® technology can be found at https://www.acontis.com/en/what-is-ethercat-communication-protocol.html.

1.2 EC-Simulator - Features

Feature ID: Unique identification used in ETG.1500 EtherCAT® MainDevice Classes

Feature name	Short description	Implemented	Feature ID
Service	Support of all commands	x	101
IRQ field in datagram	Use IRQ information from SubDevice in datagram header	x	102
SubDevices with Device Emulation	Support SubDevices with and without application controller	x	103
EtherCAT® State Machine	Support of ESM special behavior	x	104
Error Handling	Checking of network or SubDevice errors, e.g., Working Counter	–	105
VLAN	Support VLAN Tagging	–	106
EtherCAT® Frame Types	Support EtherCAT® Frames	x	107
UDP Frame Types	Support UDP Frames	–	108
Cyclic PDO	Cyclic process data exchange	x	201
Multiple Tasks	Different cycle tasks, Multiple update rates for PDO	x	202
Frame repetition	Send cyclic frames multiple times to increase immunity	x	203
Online scanning	Network configuration functionality included in EtherCAT® MainDevice	x	301
Reading ENI	Network Configuration taken from ENI file	x	–
Compare Network configuration	Compare configured and existing network configuration during boot-up	x	302
Explicit Device ID	Identification used for Hot Connect and prevention against cable swapping	x	303
Station Alias Addressing	Support configured station alias in SubDevice, i.e., enable 2nd Address and use it	x	304
Access to EEPROM	Support routines to access EEPROM via ESC register	x	305
Support Mailbox	Main functionality for mailbox transfer	x	401
Mailbox Resilient Layer	Support underlying resilient layer	x	402

continues on next page

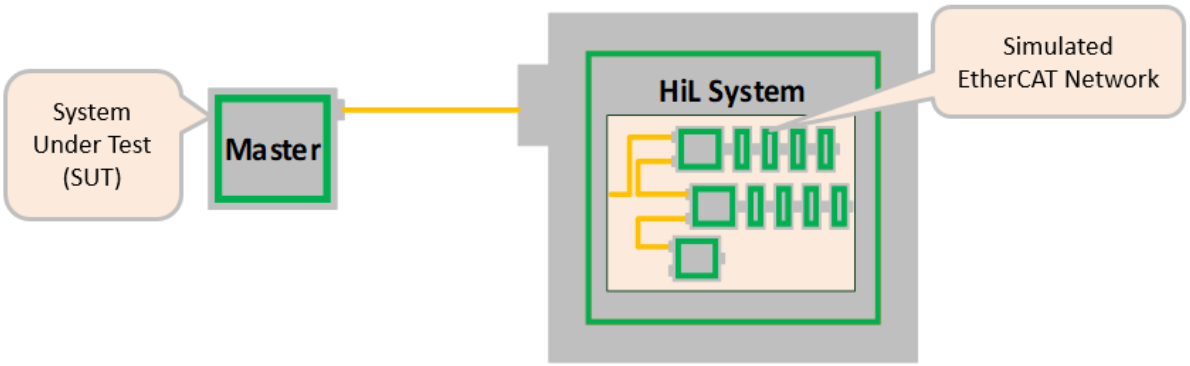
Table 1 – continued from previous page

Feature name	Short description	Implemented	Feature ID
Multiple Mailbox channels	Support multiple channels	x	403
Mailbox polling	Polling Mailbox state in SubDevices	x	404
SDO Up/Download	Normal and expedited transfer	x	501
Segmented Transfer	Segmented transfer	x	502
Complete Access	Transfer the entire object (with all sub-indices) at once	x	503
SDO Info service	Services to read object dictionary	x	504
Emergency Message	Receive Emergency messages	x	505
PDO in CoE	PDO services transmitted via CoE	–	506
EoE protocol	Services for tunneling Ethernet frames. Includes all specified EoE services	x	601
Virtual Switch	Virtual Switch functionality	–	602
EoE Endpoint to Operation Systems	Interface to the Operation System on top of the EoE layer	–	603
FoE Protocol	Support FoE Protocol	x	701
Firmware Up-/Download	Password, FileName should be given by the application	x	702
Boot State	Support Boot-State for Firmware Up/Download	x	703
SoE Services	Support SoE Services	x	801
AoE Protocol	Support AoE Protocol	x	901
VoE Protocol	External Connectivity supported	x	1001
DC support	Support of Distributed Clock	–	1101
Continuous Propagation Delay	Continuous Calculation of the compensation propagation delay	–	1102
Sync window monitoring	Continuous monitoring of the Synchronization difference in the SubDevices	–	1103
via MainDevice	Information is given in ENI file or can be part of any other network configuration. Copying of the data can be handled by MainDevice stack or MainDevice's application	x	1201

1.3 Editions (HiL / SiL)

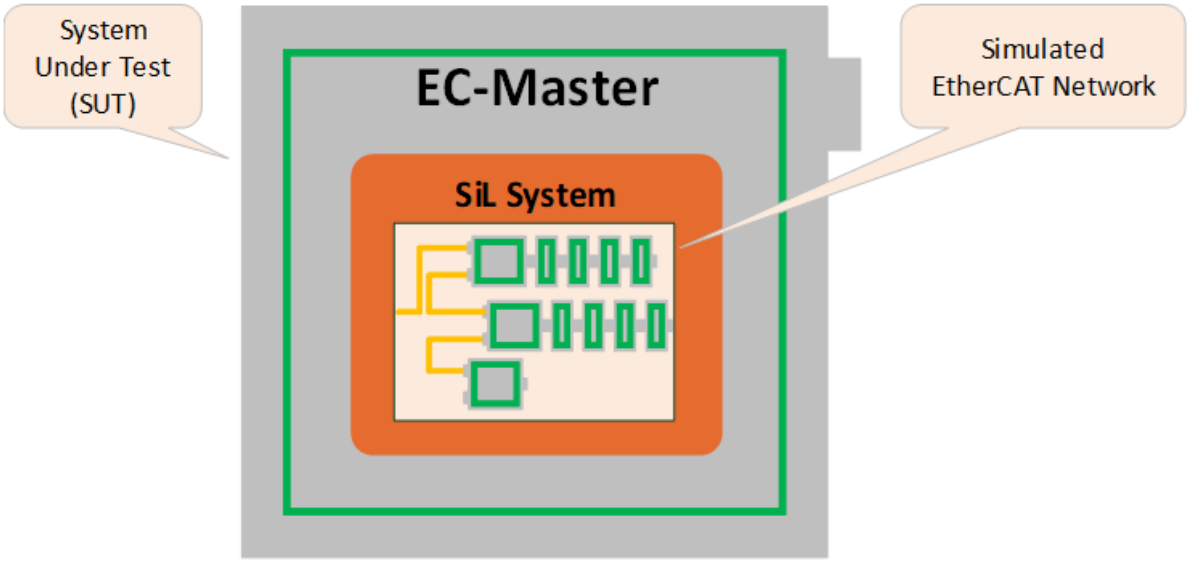
1.3.1 Hardware-in-Loop (HiL) Simulation

The System-Under-Test (SUT) is communicating via an Ethernet cable with the EC-Simulator software running on an external hardware, the HiL System. In this setup the unchanged application together with the EtherCAT® MainDevice can be tested using the standard physical network interface.



1.3.2 Software-in-the-Loop (SiL) Simulation

The EtherCAT® network is simulated by a software running on the System Under Test (SUT). Instead of communicating with the Ethernet Controller, the EC-Master is directly exchanging EtherCAT® frames with the simulation software.



1.4 Protected version

The EC-Simulator software is available in different protected versions:

Dongled
Binary with dongle protection (Linux x64, x86, Windows only)

Protected
Binary with MAC protection. Requires a physical network adapter for license check (HiL/SiL).

Unrestricted:
Binary without MAC protection

The protected version is limited to 5 SubDevices and will automatically stop after about 10 minutes of continuous operation. In order to remove this restriction a valid runtime license key or a dongle is required. The runtime license protection with license key is based on the MAC address of the Ethernet controller used for the EtherCAT® protocol. With a valid license key and Protected version the purchased features are automatically unlocked. The Dongled version needs the Dongle and WIBU service to be installed and running. The purchased features stored on the Dongle are automatically unlocked.

See also:

`esSetLicenseKey()`, *Simulator Ethernet Driver parameters*

1.4.1 Licensing procedure for Development Licenses

1. Installation of EC-Simulator Protected or Dongled version accordingly
2. In case of EC-Simulator Protected: Determine the MAC Address by calling esGetSrcMacAddress(0, &oSrcMacAddress)/ecatGetSrcMacAddress(&oSrcMacAddress) or from a sticker applied on the hardware near the Ethernet controller.
3. Send an Email with the subject "**Development License Key Request, Commission** *your commission number*" with the MAC address to sales@acontis.com
4. Acontis will create the license keys and return them in a **License Key Text File (CSV format)**.

```
Number;MAC Address;License Key
1;00-00-5A-11-77-FE;DA1099F2-15C249E9-54327FBC-3BA579B2-318AB2B6-CD13B221
2;64-31-50-80-20-4E;A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC
```

5. Activate the License Key by calling `esSetLicenseKey()` or EC_T_LINK_PARMS_SIMULATOR (SiL) with the license key that corresponds to the MAC address on the hardware and check the return code. The license key is 53 characters long. For HiL the API `esSetLicenseKey()` must be called after `esInitSimulator()` and before `esConfigureNetwork()`. For SiL the license key must be set at `EC_T_SIMULATOR_INIT_PARMS::szLicenseKey`.

   Example HiL:

```
esSetLicenseKey(0, "A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC");
```

   Example SiL:

```
$ EcMasterDemo -simulator 1 1 eni.xml --lic
↪  A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC --link -intelgbe 1 1
```

1.4.2 Licensing procedure for Runtime Licenses

1. Installation of EC-Simulator Protected or Dongled version accordingly
2. In case of EC-Simulator Protected: Determine the MAC Address by calling esGetSrcMacAddress(0, &oSrcMacAddress)/ecatGetSrcMacAddress(&oSrcMacAddress) or from a sticker applied on the hardware near the Ethernet controller.
3. Provide the MAC Addresses and numbers from **previously ordered and unused runtime license stickers** in a text file to acontis as described in the example below. Please use a separate line for each runtime license sticker number and MAC Address.

```
100-105-1-1/1603310001;00-00-5A-11-77-FE
100-105-1-1/1603310002;64-31-50-80-20-4E
```

4. Send an Email with the subject "**Runtime License Key Request, Commission** *your commission number*" with the MAC address to sales@acontis.com
5. Acontis will create the license keys and return them in a **License Key Text File (CSV format)**.

```
Number;MAC Address;License Key
1;00-00-5A-11-77-FE;DA1099F2-15C249E9-54327FBC
2;64-31-50-80-20-4E;A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC
```

6. Activate the License Key by calling `esSetLicenseKey()` or EC_T_LINK_PARMS_SIMULATOR (SiL) with the license key that corresponds to the MAC address on the hardware and check the return code. The license key is 53 characters long. For HiL the API `esSetLicenseKey()` must be called after `esInitSimulator()` and before `esConfigureNetwork()`. For SiL the license key must be set at `EC_T_SIMULATOR_INIT_PARMS::szLicenseKey`.

   Example HiL:

```
esSetLicenseKey(0, "A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC");
```

   Example SiL:

```
$ EcMasterDemo -simulator 1 1 eni.xml --lic
↪  A2B4C98D-00185EF9-22134567-DA1099F2-15C249E9-54327FBC --link -intelgbe 1 1
```

1.5 Known restrictions

Some SubDevices require extended information to be simulated.

See also:

Extended EtherCAT® Network Configuration (EXI)

The "<ExtendedInfo>"-Tag in EXI is currently not importable to EC-Engineer and must be removed before importing EXI as ENI file. Simulator settings contained in the extended information are stored in the .ecc file and can be copied between EC-Engineer instances using Copy and Paste.

Value Information (Default, Min, Max) from object dictionary cannot be retrieved using SSC, see SSC\sdoserv : „the transmission of the value info is not supported yet of the sample code".

Occasional start up error (timeout) in case of DCM Bus Shift (DC control loop not included in EC-Simulator), work-around with OsQueryMsecCount() replacement possible.

1.6 License

1.6.1 EC-Simulator license

According to EC-Simulator Software License Agreement (SLA).

1.6.2 Free Open Source Software contained in EC-Simulator

Expat XML parser license V2.6.0


```
Copyright (c) 1997-2000 Thai Open Source Software Center Ltd
Copyright (c) 2000      Clark Cooper <coopercc@users.sourceforge.net>
Copyright (c) 2000-2005 Fred L. Drake, Jr. <fdrake@users.sourceforge.net>
Copyright (c) 2001-2002 Greg Stein <gstein@users.sourceforge.net>
Copyright (c) 2002-2016 Karl Waclawek <karl@waclawek.net>
Copyright (c) 2016-2024 Sebastian Pipping <sebastian@pipping.org>
Copyright (c) 2016      Cristian Rodríguez <crrodriguez@opensuse.org>
Copyright (c) 2016      Thomas Beutlich <tc@tbeu.de>
Copyright (c) 2017      Rhodri James <rhodri@wildebeest.org.uk>
Copyright (c) 2022      Thijs Schreijer <thijs@thijsschreijer.nl>
Copyright (c) 2023      Hanno Böck <hanno@gentoo.org>
Copyright (c) 2023      Sony Corporation / Snild Dolkow <snild@sony.com>
Licensed under the MIT license:

Permission is  hereby granted,  free of charge,  to any  person obtaining
a  copy  of  this  software   and  associated  documentation  files  (the
"Software"),  to  deal in  the  Software  without restriction,  including
without  limitation the  rights  to use,  copy,  modify, merge,  publish,
distribute, sublicense, and/or sell copies of the Software, and to permit
persons  to whom  the Software  is  furnished to  do so,  subject to  the
following conditions:

The above copyright  notice and this permission notice  shall be included
in all copies or substantial portions of the Software.

THE  SOFTWARE  IS  PROVIDED  "AS  IS",  WITHOUT  WARRANTY  OF  ANY  KIND,
EXPRESS  OR IMPLIED,  INCLUDING  BUT  NOT LIMITED  TO  THE WARRANTIES  OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
NO EVENT SHALL THE AUTHORS OR  COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR  OTHER LIABILITY, WHETHER  IN AN  ACTION OF CONTRACT,  TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
USE OR OTHER DEALINGS IN THE SOFTWARE.
```

1.6.3 Free Open Source Software supported by EC-Simulator

The following components are not part of EC-Simulator, but relate to it:

acontis atemsys Linux kernel module

The acontis atemsys is licensed under the GPL:

```
Copyright (c) 2009 - 2020 acontis technologies GmbH, Ravensburg, Germany
All rights reserved.

This program is free software; you can redistribute  it and/or modify it
under  the terms of  the GNU General  Public License as published by the
Free Software Foundation;  either version 2 of the  License, or (at your
option) any later version.
```

1.7 Versioning

EC-Simulator follows the following versioning scheme: **V***MAJOR***.** *MINOR***.** *SERVICEPACK***.** *BUILD* (e.g. V3.2.1.04).

The libraries are binary compatible by unchanged *MAJOR* and *MINOR* digits. If *SERVICEPACK* increments, *BUILD* restarts with 01. *BUILD* 99 is reserved for test builds that have not been officially released for productive usage.

2 Getting Started

2.1 EC-Simulator Architecture

The EC-Simulator library is implemented in C++ and can be easily ported to any embedded OS platforms using an appropriate C++ compiler. The API functions are C language interfaces, thus the stack can be used in ANSI-C as well as in C++ environments.

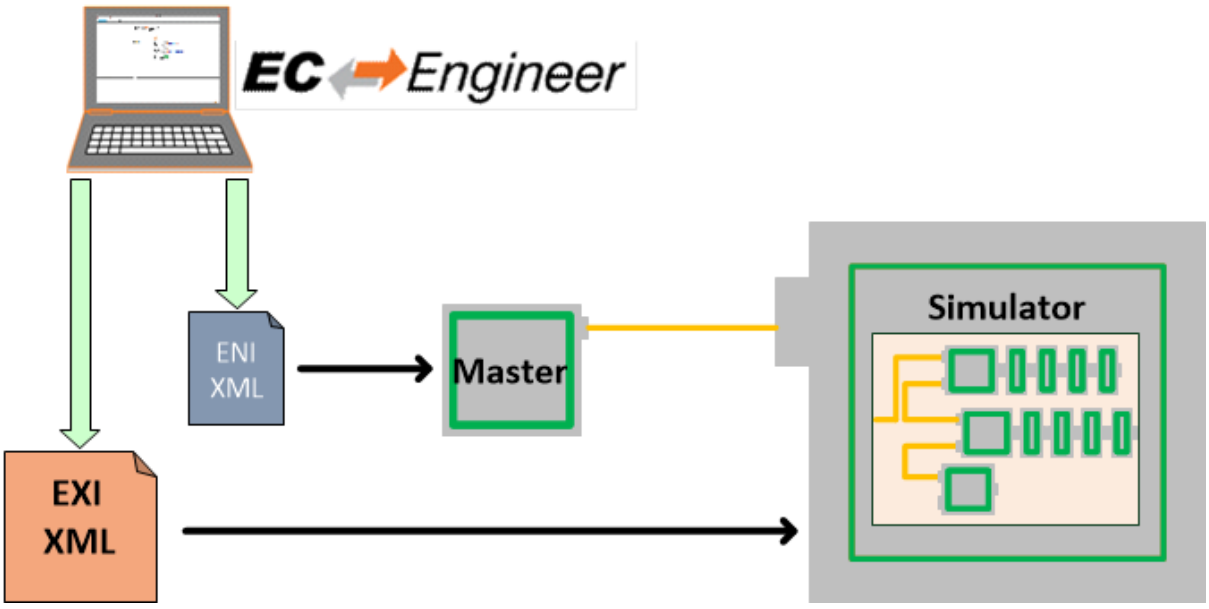
There are two different architecture editions as described below. The architectures contain the following:

- EC-Simulator Stack Core: In the core module cyclic (process data update) and acyclic (mailbox) EtherCAT® commands are sent and received. Among others there exist some state machines to handle for example the mailbox protocols.
- Configuration Layer: The EC-Simulator stack is configured using an XML file whose format is fixed in the EtherCAT® specification ETG.2100 (ENI) with extended information (EXI). The EC-Simulator contains an OS independent XML parser.
- Real-time Ethernet Driver Layer: This layer exchanges Ethernet frames between the MainDevice and the Simulator. If hard real-time requirements exist, this layer has to be optimized for the network adapter card in use.
- OS Layer: All OS dependent system calls are encapsulated in a small OS layer. Most functions are that easy that they can be implemented using simple C macros.

2.2 Extended EtherCAT® Network Configuration (EXI)

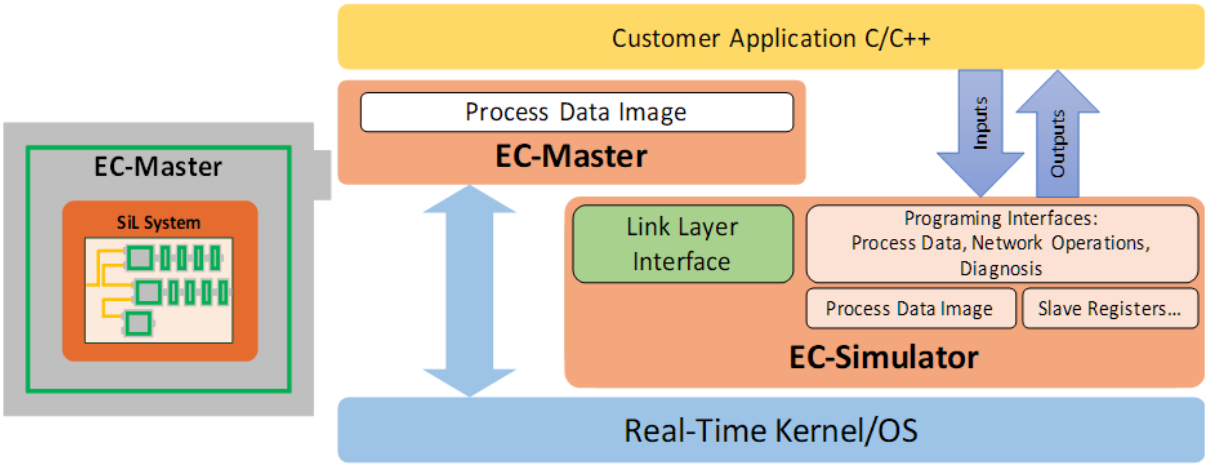
The EC-Simulator stack has to know about the EtherCAT® bus topology and the cyclic/acyclic frames to exchange with the SubDevices. This configuration is determined in a configuration file which has to be available in the EtherCAT® Network Information Format (ENI). This format is completely independent from EtherCAT® SubDevice vendors, from MainDevice stack vendors and from EtherCAT® configuration tools. Thus inter-operability between those vendors is guaranteed.

Because the ENI file does not include information about the SubDevice hardware, the EXI (EtherCAT® Extended Information) file includes EEPROM content and SubDevice Object Dictionary from ESI and ESC register values from real network is typically needed. The EXI file can be exported using EC-Engineer.



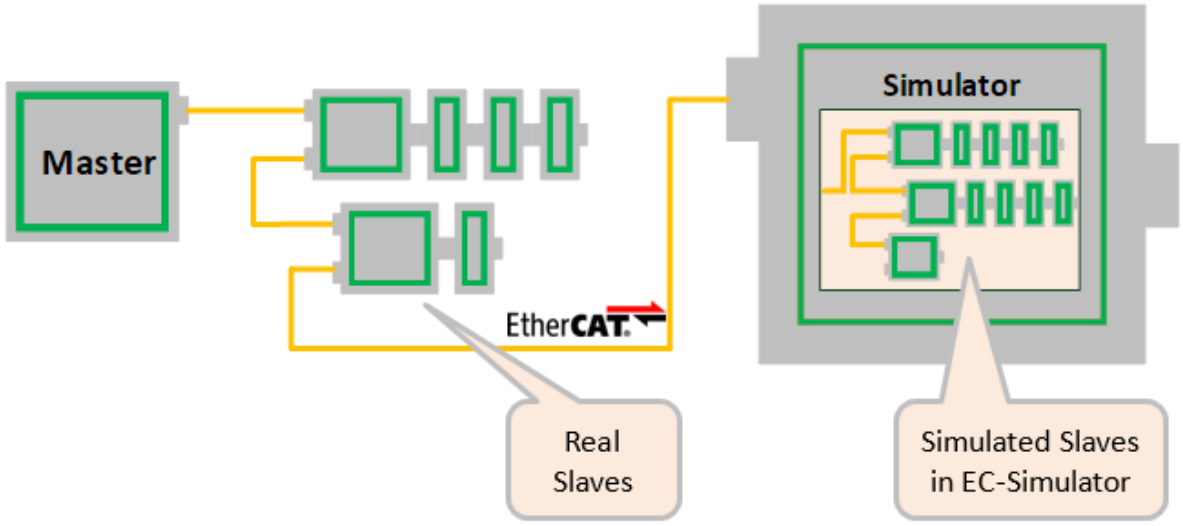
2.3 SiL Simulation Software architecture

The SiL Simulation Software architecture does not need a physical network interface for SubDevice simulation. The EC-Simulator is loaded as Ethernet Driver instead to simulate the EtherCAT® Network on a System controller running the EC-Master.

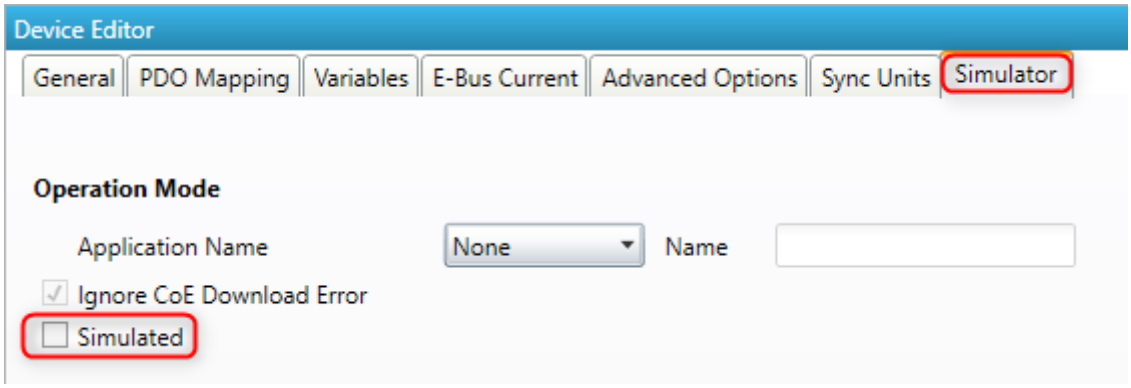


2.4 Mixed mode of real SubDevices and simulated SubDevices

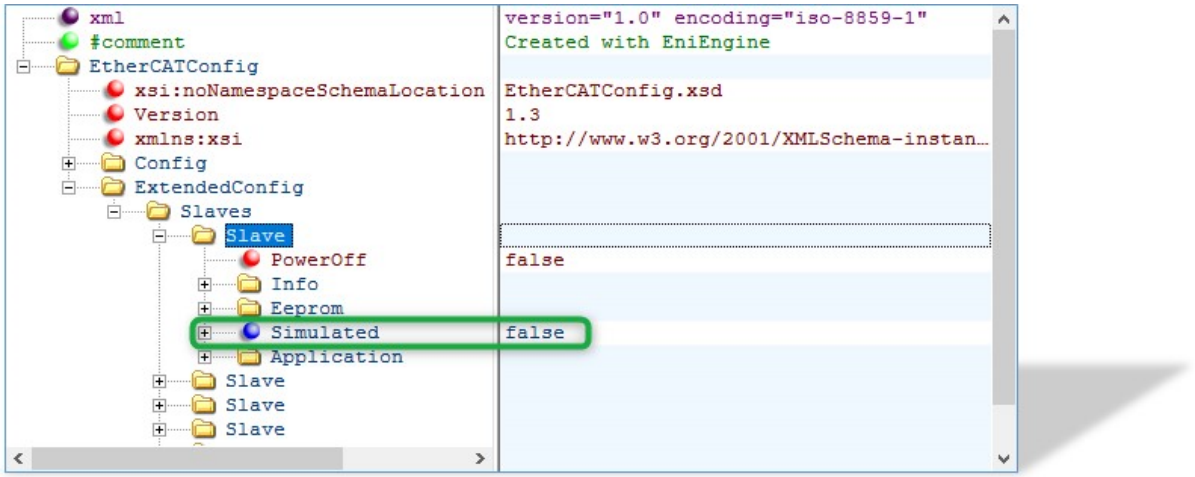
In order to implement restbus simulation it is possible to mix real SubDevices and simulated SubDevices like this:



All real SubDevices must be configured accordingly:



See also /EtherCATConfig/ExtendedConfig/Slaves/Slave/Simulated in the EXI file:



The EC-Simulator will automatically assign the network adapters to the corresponding ports in order of configuration.

See also:

EC_T_SIMULATOR_INIT_PARMS::apLinkParms at *esInitSimulator()*.

2.5 Operating system configuration

The main task is to setup the operating system to support the appropriate network adapter for EtherCAT® usage and for some systems real-time configuration may be needed.

The operating system-specific settings and configurations are described in *Platform and Operating Systems (OS)*.

2.6 Using emllSimulator (SiL)

The EcSimulatorSilDemo example program as well as all EC-Master example programs like EcMasterDemo, EcMasterDemoDc, etc. support the EC-Simulator (SiL) Ethernet Driver emllSimulator. In this case no physical Real-time Ethernet Driver is passed to the EC-Master. Instead the emllSimulator (SiL) simulates the physical network. The command line syntax is for example:

```
$ EcSimulatorSilDemo -simulator 1 1 <exi-file> -f <eni-file> -t 0 -sp
```

The EC-Simulator Instance ID when calling EC-Simulator APIs is the Real-time Ethernet Driver Instance ID given at command line:

```
$ EcSimulatorSilDemo -simulator <Instance-ID> 1 exi.xml -f exi.xml -t 0 -sp
```

Valid values for Instance ID of emllSimulator are 1 … 24 .

emllSimulator is dynamically loaded within ecatInitMaster() . In contrast to HiL, in case of SiL, ecatInitMaster() automatically calls esInitSimulator(), so the EC-Simulator SiL application may not call esInitSimulator() for the given Instance ID.

The EC-Simulator APIs can be used in the EcMasterDemo, too. The APIs are declared in SDK\INC\EcSimulator.h and EcSimulator.lib / libEcSimulator.so must be linked to the application to import the EC-Simulator APIs.

The EC-Simulator SiL Protected version with MAC based protection needs the corresponding Real-time Ethernet Driver settings provided.

See also:

Protected version

2.6.1 Simulator Ethernet Driver parameters

The parameters to the Simulator Ethernet Driver are setup-specific. The function `CreateLinkParmsSimulator()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Ethernet Driver instance.

struct **EC_T_LINK_PARMS_SIMULATOR**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_SIMULATOR.

EC_T_OS_PARMS ***pOsParms**
OS layer parameters

EC_T_CNF_TYPE **eCnfType**
Optional: create slaves from ENI/EXI, see esConfigureNetwork

EC_T_BYTE ***pbyCnfData**
Optional: create slaves from ENI/EXI, see esConfigureNetwork

EC_T_DWORD **dwCnfDataLen**
Optional: create slaves from ENI/EXI, see esConfigureNetwork

EC_T_SIMULATOR_DEVICE_CONNECTION_DESC **oDeviceConnection**
See EC_SIMULATOR_DEVICE_CONNECTION_TYPE…

EC_T_BOOL **bConnectHcGroups**
Connect hot connect groups in topology (floating group heads to free ports)

EC_T_DWORD **dwSimulatorAddress**
Reserved

EC_T_DWORD **dwBusCycleTimeNsec**
Cycle time of simulator job task

EC_T_BOOL **bDisableProcessDataImage**
Don't allocate Process Data Image at simulator (legacy support, CiA402 simulation)

EC_T_UINT64 **qwOemKey**
64 bit OEM key (optional)

EC_T_BYTE **abyMac**[6]
MAC address

EC_T_DWORD **dwRxBufferCnt**
Frame buffer count for IST

EC_T_BOOL **bJobsExecutedByApp**
EC_FALSE: esExecJob explicitly called by application, EC_TRUE: implicitly by emllSimulator

EC_T_CHAR **szLicenseKey**[EC_SIMULATOR_KEY_SIZE]
License key (zero terminated string)

EC_T_LINK_PARMS ***apLinkParms**[EC_SIMULATOR_MAX_LINK_PARMS]
Link parameters of network adapters passed to EC-Simulator Core, e.g. for validation of MAC address of license key

EC_T_WORD **wRasServerPort**
RAS server port

EC_T_CPUSET **oRasCpuAffinityMask**
RAS server threads CPU affinity mask

EC_T_DWORD **dwRasPriority**
RAS server threads priority

EC_T_DWORD **dwRasStackSize**
RAS server threads stack size

EC_T_PERF_MEAS_INTERNAL_PARMS **PerfMeasInternalParms**
Internal performance measurement parameters

emllSimulator accepts optional parameters, e.g.:

```
$ -simulator 1 1 <exi-file> [--mac <address>] [--lic <key>] [--link <link-parms>]
↪  [--sp] [--connect <type> <id> <address> <port>]
```

2.7 EC-Simulator Software Development Kit (SDK)

The EC-Simulator SDK is needed to write applications based on the EC-Simulator stack. The EC-Simulator stack is shipped as a library which is linked together with the application.

The following components are supplied together with an SDK:

- ….Bin: Executables containing the EC-Simulator stack
- ….Doc: Documentation
- ….Examples: Example application(s)
- **Libraries and header files to build C/C++-applications.**
  - ….\SDK\INC: header files to be included with the application
  - ….\SDK\LIB: libraries to be linked with the application
- ….\Sources\Common: Shared C++-files

For all operating systems the same principal rules to generate the example applications can be used.

Include search path

The header files are located in the following directories:

1. <InstallPath>\Examples\<ExampleName> (whith e.g. <ExampleName>=EcSimulatorHilDemo)
2. <InstallPath>\Examples\Common\<OS> (where <OS> is a placeholder for the operating system)
3. <InstallPath>\Examples\Common
4. <InstallPath>\SDK\INC\<OS> (where <OS> is a placeholder for the operating system)
5. <InstallPath>\SDK\INC
6. <InstallPath>\Sources\Common

Preprocessor macro

The demo applications are the same for all operating systems. The appropriate pre-processor macro has to be set for the operating system.

See also:

OS Compiler settings

Libraries

The libraries located in <InstallPath>\SDK\LIB\<OS>\<ARCH> have to be added (<OS> is a placeholder for the operating system used and <ARCH> for the architecture if different architectures are supported).

2.7.1 OS Compiler settings

Linux Compiler settings

The following settings are necessary to build the example application for Linux.

- Possible ARCHs (see ATECAT_ARCHSTR in `SDK/INC/Linux/EcOsPlatform.h`):

aarch64 (ARM 64Bit), armv7-vfp-eabihf (ARM 32Bit), PPC (PPC 32Bit with “-te500v2”), riscv64 (RISC-V 64Bit), x64 (x86 64Bit), x86 (x86 32Bit)

An ARM VFP system returns success on “readelf -A /proc/self/exe | grep Tag_ABI_VFP_args”. If “readelf” isn’t available on the target, the matching ARM version can be figured out by trying to run EcSimulatorHilDemo.

- **Extra include paths:**
  <InstallPath>/Examples/EcSimulatorHilDemo <InstallPath>/SDK/INC/Linux <InstallPath>/SDK/INC <InstallPath>/Sources/Common
- **Extra source paths:**
  <InstallPath>/Examples/EcSimulatorHilDemo <InstallPath>/Sources/Common/EcTimer.cpp
- **Extra library paths to the main EtherCAT® components (replace “x86” according to ARCH):**
  <InstallPath>/SDK/LIB/Linux/x86
- **Extra libraries (in this order)**
  EcSimulatorRasServer EcSimulator pthread dl rt

Microsoft Windows

The following settings are necessary to build the example application for Windows:

- **Library path of the main EtherCAT® components:**
  <InstallPath>/SDK/LIB/Windows/<Arch>
- **Include path:**
  <InstallPath>/SDK/INC/Windows <InstallPath>/SDK/INC <InstallPath>/Sources/Common

3 Software Integration

3.1 EcSimulatorSilDemo

The example application EcSimulatorSilDemo will handle the following tasks:

- Stack initialization
- Flash process data
- Out of the box solution for different operating systems:
  - Windows
  - Linux
  - …
- Example implementation for polled mode operation
- Thread with periodic tasks and application thread already implemented
- The output messages of the demo application will be printed on the console as well as in log files (*.log).

3.1.1 File reference

The Demo application consists of the following files:

EcDemoMain.cpp	Entry point for the different operating systems
EcDemoPlatform.h	Operating system specific settings (task priorities, timer settings)
EcDemoApp.cpp	Initialize, start and terminate the MainDevice and EC-Simulator stack
EcDemoApp.h	Application specific settings for EcSimulatorSilDemo
EcDemoParms.cpp	Parsing of command line parameters and generic helper functions
EcDemoParms.h	Basic configuration parameters
EcSelectLinkLayer.cpp	Common functions which abstract the command line parsing into Real-time Ethernet Driver parameter
EcNotification.cpp	SubDevice monitoring and error detection (function `NotifyWrapper()`)
EcSlaveInfo.cpp	SubDevice information services
EcLogging.cpp	Message logging functions
EcTimer.cpp	Start and monitor timeouts
EcDemoTimingTask.cpp	Operating system independent default timing task implementation (base class)
EcLogging.cpp	Message logging functions
EcSdoServices.cpp	CoE object dictionary example
EcTimer.cpp	Start and monitor timeouts

3.1.2 Logging

The example program demonstrates how messages can be processed by the application, see `Examples/Common/EcLogging.cpp`.

The messages handled by `EcLogging.cpp` are of different type, e.g. MainDevice and EC-Simulator stack Log Messages and Application Messages are logged to console and/or to files. The verbosity of the demo given as console parameter "-v" is used to determine the log level of the application, see "set application log level" in `EcDemoMain.cpp`. Logging is configured on MainDevice stack initialization. The MainDevice and EC-Simulator stack automatically filters log messages according to the log level. Log messages which are less relevant according to their severity (see EC_LOG_LEVEL…) are filtered out.

CAtEmLogging has various parameters beside the log level, like Roll Over setting, log task prio and affinity, log buffer size, etc. See InitLogging in `EcLogging.h`, `EcDemoMain.cpp` for reference.

The application can override CAtEmLogging::PrintConsole / CAtEmLogging::PrintMsg if the default handler in `EcLogging.cpp` does not fulfill the application's needs.

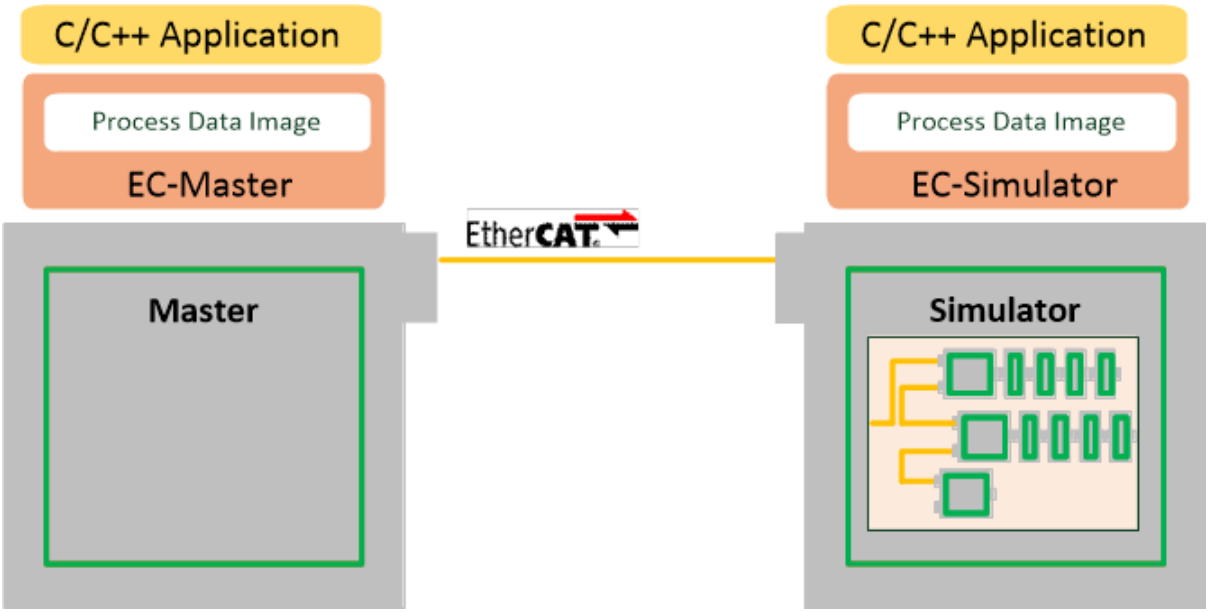
Important: The callback is typically called from the Job Task's context and should return as fast as possible.

Important: Logging to files is disabled by default for some OS, because e.g. a file system must be added explicitly. Setting bLogFileEnb to 1 is needed for some operating systems to enable file logging.

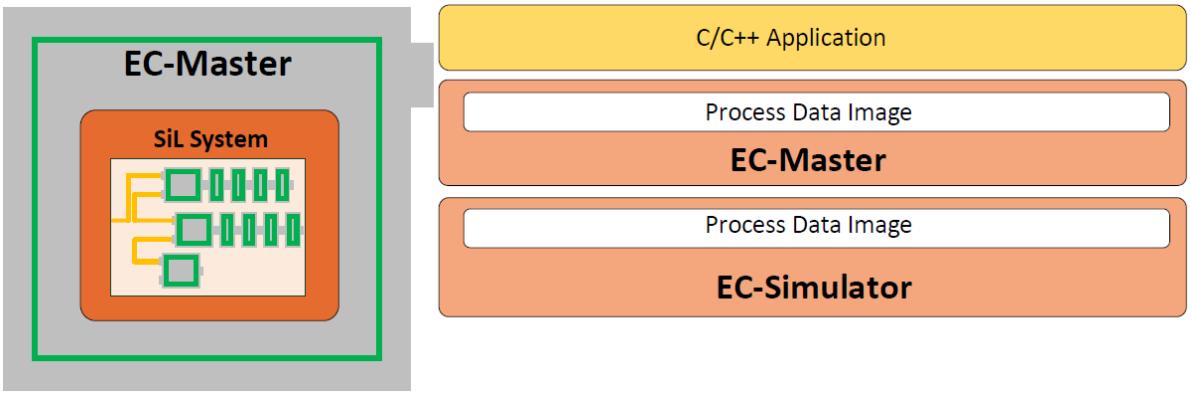
Identical messages are skipped automatically by default.

3.2 Process data update and synchronization

The EtherCAT® MainDevice cyclically sends frames containing the outputs of the Process Data Image to the EtherCAT® network and the Simulator synchronizes the Process Data Image with the frames from the MainDevice and sends the response back to the MainDevice. The MainDevice cyclically receives the inputs of the Process Data Image from the Simulator.

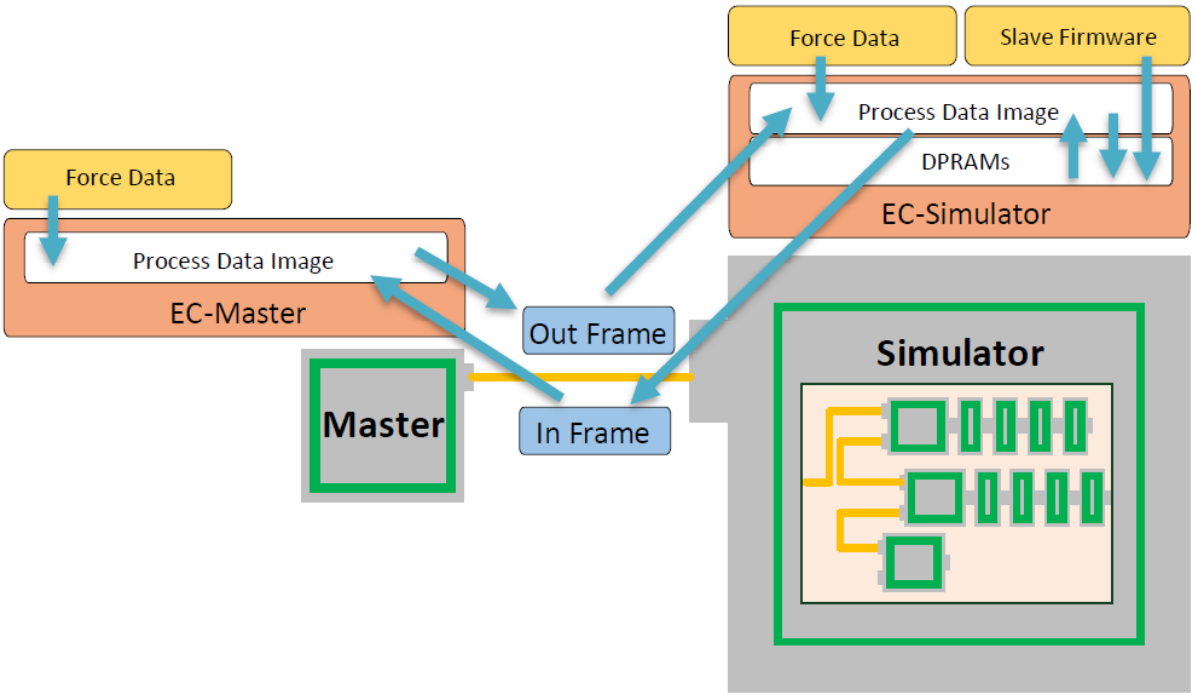


The SiL Application directly accesses the Process Data Image at MainDevice and Simulator:



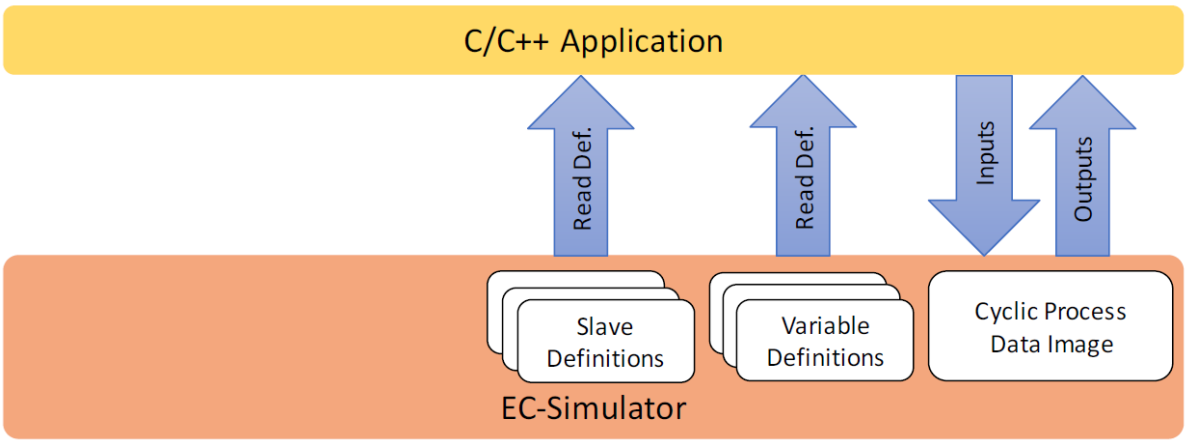
3.2.1 Forced Data, Firmware Data, DPRAM

The EC-Simulator stack synchronizes the outputs and inputs from the Process Data Image and the SubDevice DPRAM areas containing process data objects individually.



3.2.2 EC-Simulator stack as process data memory provider

By default the EC-Simulator stack internally allocates the memory needed to store input and output process data values.



In order to update process data, the application's JobTask has to call:

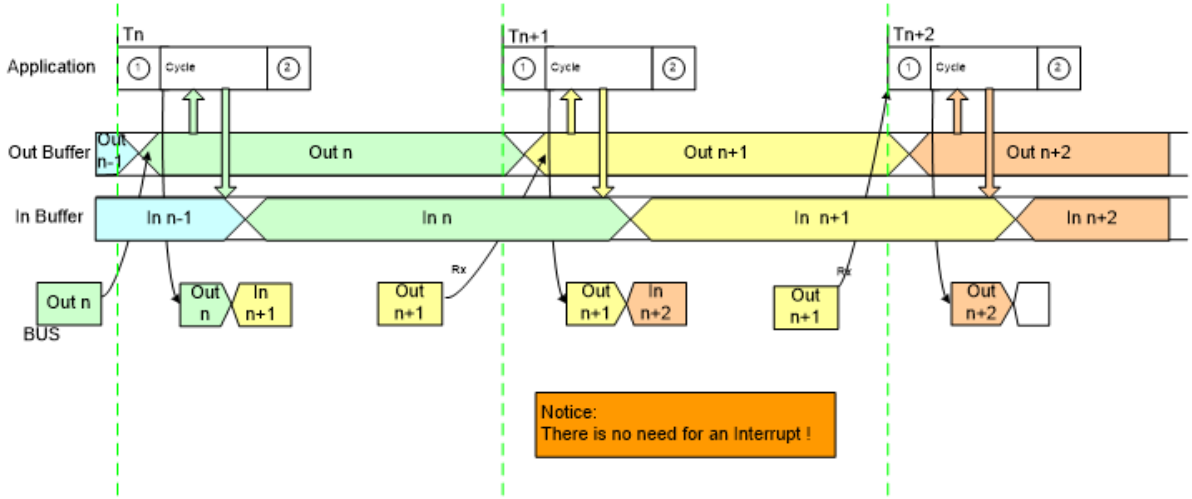
- **eUsrJob_ProcessAllRxFrames (polling mode only)**
  Process all received frames from Real-time Ethernet Driver
- **eUsrJob_SimulatorTimer**
  Trigger state machines

Important: In polling mode, the MainDevice cycle time must be at least two times higher than the simulator cycle time. E.g. if the simulator runs with 1 ms, the MainDevice cycle time must be at least 2 ms. If the Real-time Ethernet Driver is running in interrupt mode (non-standard), processing of received frames is done immediately after the frame is received.

See also:

EC_T_SIMULATOR_INIT_PARMS::bDisableProcessDataImage at *esInitSimulator()* (HiL) or *EC_T_LINK_PARMS_SIMULATOR::bDisableProcessDataImage* (SiL) for how to disable allocation and usage of the process data image.

Cyclic frames - Real-time Ethernet Driver in polling mode



Application has to perform:

```
/* Job 1: incoming process data is stored to Process data image (polling mode␣
↪only) */
esExecJob(dwSimulatorInstanceId, eUsrJob_ProcessAllRxFrames, &bPrevCycProcessed);
...
...
/* do process data cycle */
...
...
/* Job 2: trigger state machines, which are necessary to perform any status change␣
↪or internal administration tasks */
esExecJob(dwSimulatorInstanceId, eUsrJob_SimulatorTimer, EC_NULL);
```

For closer details find an example project “EcSimulatorHilDemo” in the folder “Examples”.

3.3 Accessing process data in the application

The process data, exchanged between the EC-Simulator stack and the SubDevices in every cycle, are stored in the process data image. There are two separate memory areas, one for the input data and another one for the output data. The base addresses of these areas are provided by calling the functions `esGetProcessImageInputPtr()` and `esGetProcessImageOutputPtr()`. The size of the process data image is defined in the ENI file under "EtherCATConfig/Config/ProcessImage/Inputs/ByteSize" and "EtherCATConfig/Config/ProcessImage/Outputs/ByteSize" and is returned by *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE* and `esRegisterClient()` at `EC_T_REGISTERRESULTS::dwPDOutSize` and `EC_T_REGISTERRESULTS::dwPDInSize`.

See also:

- `esGetProcessImageOutputPtr()`
- `esGetProcessImageInputPtr()`
- *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- `esRegisterClient()`: `EC_T_REGISTERRESULTS::dwPDOutSize`, `EC_T_REGISTERRESULTS::dwPDInSize`

3.3.1 Process Data Access Functions selection

Process data variables that are packed as an array of bits are bit aligned and not byte aligned in the process data.

See also:

EC_COPYBITS for how to copy data areas with bit offsets that are not byte aligned.

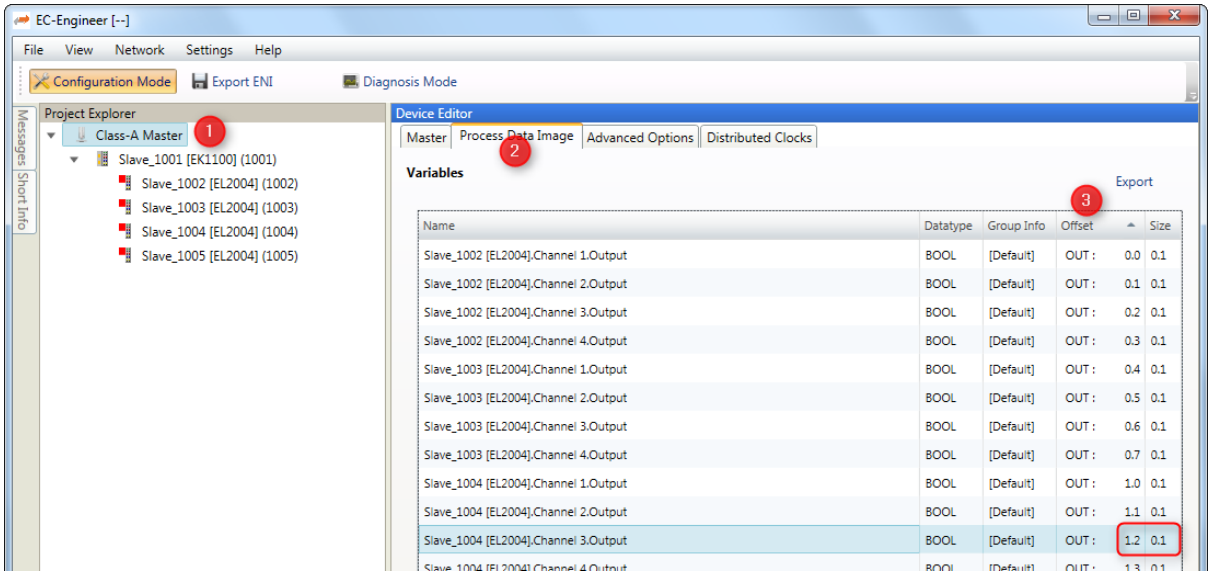
Getting and setting bits that are bit aligned and not byte aligned should be done using EC_SETBITS and EC_GETBITS. Complete `EC_T_BYTE`, `EC_T_WORD`, `EC_T_DWORD`, `EC_T_QWORD` can be accessed more efficiently using the appropriate macros according to the following table.

Note that these functions do not initiate any transfer on wire. Typically process data is transferred as little endian on wire and therefore must be swapped on big endian systems like PPC to be correctly interpreted, see hints in table below.

Variable type (Bit size)	**Macro**	**Hint**
EC_T_BYTE (8)	N/A	Bytes are byte-aligned and can be directly addressed at pbyBuffer[BitOffset/8]
EC_T_WORD (16)	EC_SET_FRM_WORD, EC_GET_FRM_WORD	Contains swap for big endian systems
EC_T_DWORD (32)	EC_SET_FRM_DWORD, EC_GET_FRM_DWORD	Contains swap for big endian systems
EC_T_QWORD (64)	EC_SET_FRM_QWORD, EC_GET_FRM_QWORD	Contains swap for big endian systems
Bit(1)	EC_SETBITS / EC_GETBITS	

3.3.2 Process variables' offset and size

The following screenshot shows variables' offset and size within the Process Data Image:



Accessing the process data of a specific SubDevice always works by adding an offset to the base address.

There are different ways possible to get this offset. All offsets are given as bit offsets! The offset values will not change until a new configuration is provided (s.a. *esConfigureNetwork()*), therefore it is sufficient to load them once right after *esConfigureNetwork()*, it is not needed every cycle.

Manually hard coded offsets (compiled in application)

The offset value is figured out from the EtherCAT® configuration tool. It's not recommended to use fixed values because the offsets change when adding/removing SubDevices to/from the configuration.

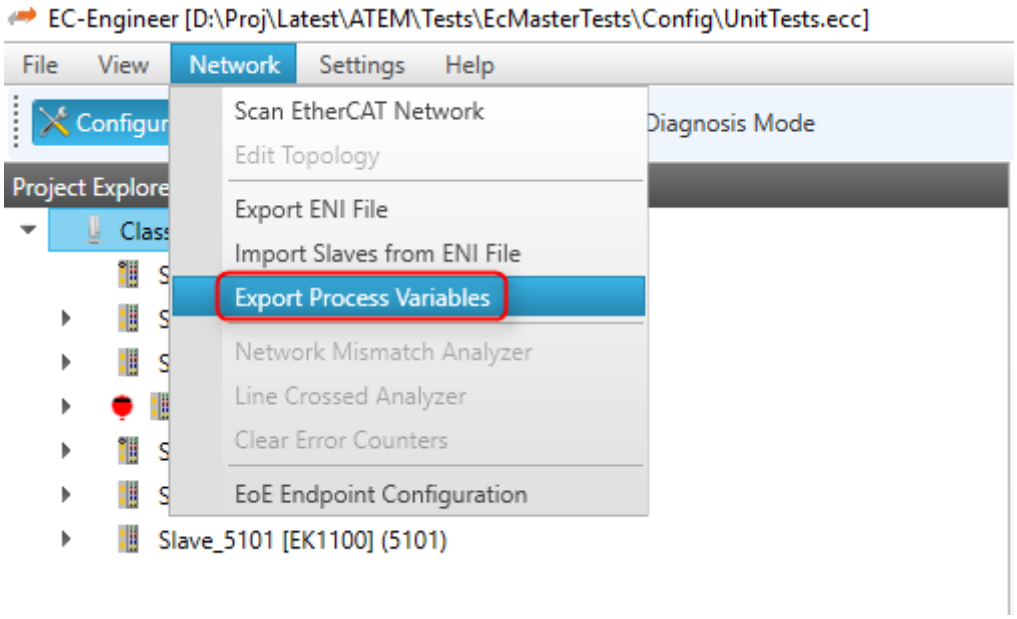
As listed in the screenshot above "SubDevice_1004 [EL2004].Channel 3.Output" in the example is at offset 1.2 with size 0.1. The numbering is Byte.Bit so the offset in the example is Byte 1, Bit 2 means bit offset 8*1+2 = 10 and size is 0*8+1 = 1.

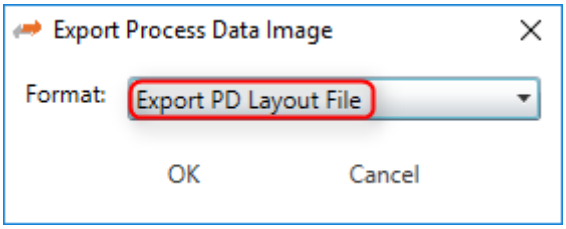
Sample code:

```
EC_T_BYTE byNewValue = 0x01;

/* get variable from process data */

EC_GETBITS(esGetProcessImageOutputPtr(dwSimulatorInstanceId), &byNewValue, 10 /*␣
↪offset in bits */, 1 /* size in bits */);
```

Generated PD Layout (compiled in application)

The EC-Engineer can export the process variables to a PD Layout File (C-Header) using the menu item "Network > Export Process Variables" as shown in the following screenshots:





This will generate a header file containing the SubDevice's variables like this:

```
    [...]
#include EC_PACKED_INCLUDESTART(1)
#define PDLAYOUT_OUT_OFFSET_SLAVE_2002 22
typedef struct _T_PDLAYOUT_OUT_SLAVE_2002
{
    EC_T_SWORD  swChannel_1_Output; // Slave_2002 [EL4132].Channel 1.Output ...
    EC_T_SWORD  swChannel_2_Output; // Slave_2002 [EL4132].Channel 2.Output ...
} EC_PACKED(1) T_PDLAYOUT_OUT_SLAVE_2002;
#include EC_PACKED_INCLUDESTOP
```

Example for changing values in e.g. myAppWorkPd:

```
EC_T_WORD wVal = EC_GET_FRM_WORD(&((T_PDLAYOUT_OUT_SLAVE_2002*)(pbyPDOut +
PDLAYOUT_OUT_OFFSET_SLAVE_2002))->swChannel_1_Output);
```

SubDevice / variable offset from configuration

esGetCfgSlaveInfo

Figure out the SubDevice offsets dynamically by calling the function *esGetCfgSlaveInfo()*: The offsets are stored in *EC_T_CFG_SLAVE_INFO::dwPdOffsIn* and *EC_T_CFG_SLAVE_INFO::dwPdOffsOut*. E.g. setting "SubDevice_1004 [EL2004].Channel 3.Output" according to the screenshot above is like:

```
EC_T_BYTE byNewValue = 0x01;

EC_T_CFG_SLAVE_INFO SlaveInfo;
```

(continues on next page)

(continued from previous page)

```
dwRes = esGetCfgSlaveInfo(dwSimulatorInstanceId, EC_TRUE, 1004, &SlaveInfo);

/* get variable from process data */
EC_GETBITS(esGetProcessImageOutputPtr(dwSimulatorInstanceId), &byNewValue,␣
↪SlaveInfo.dwPdOffsOut + 2 /* variable relative offset in bits within SubDevice␣
↪sync unit */, 1 /* variable size in bits */);
```

esFindInpVarByName

Figure out the variable offset by calling the function *esFindInpVarByName()* or *esFindOutpVarByName()*: The offset is stored in *EC_T_PROCESS_VAR_INFO::nBitOffs*. Each input or output has a unique variable name. All variable names are stored in the ENI file under "EtherCATConfig/Config/ProcessImage/Inputs/Variable". E.g. setting "SubDevice_1004 [EL2004].Channel 3.Output" according to the screenshot above is like:

```
EC_T_BYTE byNewValue = 0x01;

EC_T_PROCESS_VAR_INFO VarInfo;
dwRes = esFindOutpVarByName(dwSimulatorInstanceId, "SubDevice_1004 [EL2004].
↪Channel 3.Output", &VarInfo)

/* get variable from process data */
EC_GETBITS(esGetProcessImageOutputPtr(dwSimulatorInstanceId), &byNewValue, VarInfo.
↪nBitOffs /* variable absolute offset in bits within Process Data Image */,␣
↪VarInfo.nBitSize /* size in bits */);
```

3.4 Hot Connect

The EC-Simulator supports Hot Connect. The initial presence of Hot Connect groups can be configured. If bConnectHcGroups is set, all Hot Connect groups will be initially connected, else only the mandatory SubDevices will be initially connected.

See also:

- HiL: *EC_T_SIMULATOR_INIT_PARMS::bConnectHcGroups* at *esInitSimulator()*
- SiL *EC_T_LINK_PARMS_SIMULATOR::bConnectHcGroups* at ecatInitMaster
- *esPowerSlave()* , *esDisconnectPort()* , *esConnectPorts()*

3.4.1 Configured Station Alias

The network configuration contains the initial setting of the Configured Station Alias e.g. used by the MainDevice to identify Hot Connect Group Heads.

In order to change the Configured Station Alias after loading the configuration, the EEPROM must be updated and the SubDevice needs to be power cycled.

The following example demonstrates this:

Configured Station Alias Update Example

```
    EC_T_WORD wSlaveAddress = 1001;
    EC_T_WORD wAlias = 1234;
    EC_T_WORD wEepromVal = EC_GET_FRM_WORD(&wAlias);

    /* write new station alias to EEPROM */
    dwRes = esWriteSlaveEEPRom(dwSimulatorId, EC_TRUE, wSlaveAddress, ESC_SII_REG_
↪ALIASADDRESS, &wEepromVal, 1, EEPROM_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }

    /* power cycle slave to apply station alias from ESC_SII_REG_ALIASADDRESS␣
↪(EEPROM) to ECREG_STATION_ADDRESS_ALIAS (ESC register 0x0012) */
    dwRes = esPowerSlave(dwSimulatorId, wSlaveAddress, EC_FALSE);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
    dwRes = esPowerSlave(dwSimulatorId, wSlaveAddress, EC_TRUE);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
```

3.5 Cable Redundancy

The EC-Simulator supports Cable Redundancy by using two Network Interfaces to provide the Redundancy feature. Frames are sent from and received on both interfaces.

Important: The MainDevice has restrictions in case of cable redundancy, e.g. it is not allowed to use LRW EtherCAT® commands.

The EC-Simulator SiL cannot automatically connect the red adapter to a matching port within the simulated network topology. The connection must be set explicitly at *EC_T_LINK_PARMS_SIMULATOR::oDeviceConnection*.

3.6 Running EcSimulatorSilDemo with Cable Redundancy

3.6.1 Linux (EcSimulatorSilDemo)

- Starting EcSimulatorSilDemo with two links configured and red adapter connected to simulator instance 1, SubDevice 1001, Port C:

```
$ cd /opt/EC-Simulator/Bin/Linux/x64
$ LD_LIBRARY_PATH=. ./EcSimulatorSilDemo -f exi.xml -simulator 1 1 exi.xml
↪  -simulator 2 1 exi.xml --connect slave 1 1001 2
```

3.6.2 EcMasterDemoPython (SiL)

- Script arguments for `Examples/EcMasterDemoPython/EcDemoApp.py` with two emllSimulator instances configured and red adapter connected to simulator instance 1, SubDevice 1001, Port C:

```
$ -mode 1 -file exi.xml -time 20000 -lvl 3 -link "simulator exi.xml 1 1" -linkred
↪  "simulator exi.xml 2 1" -simconnect "slave 1 1001 2"
```

3.7 Error detection and diagnosis

The EC-Simulator API functions generally return *EC_E_NOERROR* or an error code.

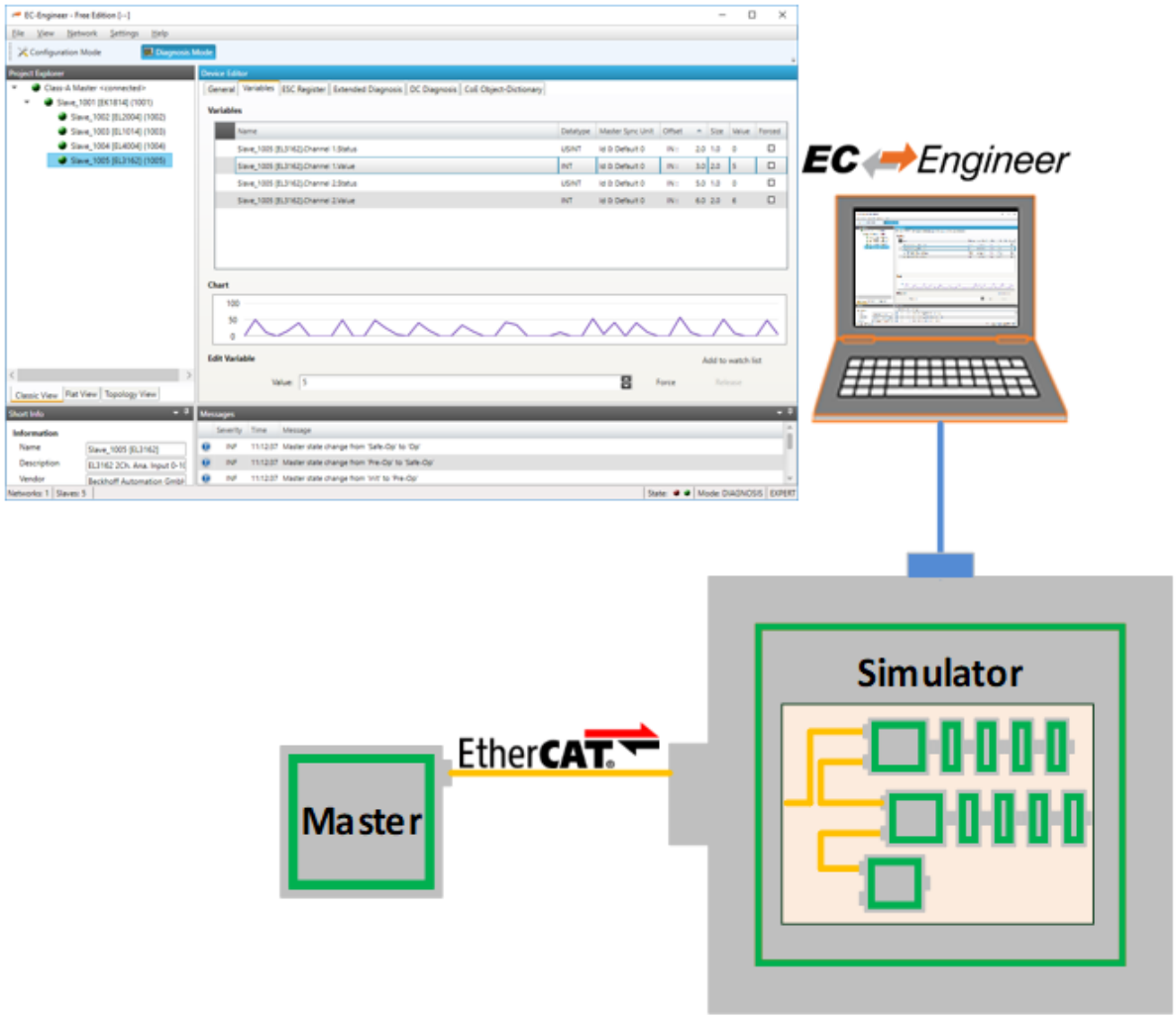
Messages are logged as described in *esInitSimulator()*.

Important: Logging is typically done from within the JobTask context so the handler should queue the messages and process them in a low priority task. See logging in EcSimulatorSilDemo .

3.8 RAS-Server for EC-Lyser and EC-Engineer

3.8.1 Integration Requirements

To use the diagnosis tool EC-Lyser with a customer application, some modifications have to be done during integration of the EC-Simulator. The task is to integrate and start the Remote API Server system within the custom application, which provides a socket based uplink, which later on is connected by the EC-Lyser.



An example on how to integrate the Remote API Server within the application is given with the example application EcSimulatorHilDemo, which is preconfigured to listen for EC-Lyser on TCP Port 6000 when command line parameter “-sp” is given.

To demonstrate the steps needed within a custom application, a developer may use the following pseudo-code segment as a point of start. The Remote API Server library “EcSimulatorRasServer” must be linked.

3.8.2 Pseudo Code

```
#include "EcSimulatorRasServer.h"

/* custom Remote API Notification handler, example in EcSimulatorHilDemo␣
↪(EcNotification.cpp) */
EC_T_DWORD RasNotifyCallback(
    EC_T_DWORD           dwCode,     /**< [in]   Notification code identifier */
    EC_T_NOTIFYPARMS*    pParms      /**< [in]   Notification data portion */
                        )
{
    /* custom notification handler */
}
```

```
    /* initialization  */
    EC_T_RAS_SERVER_PARMS  oRemoteApiConfig;
```

(continues on next page)

(continued from previous page)

```
    OsMemset(&oRemoteApiConfig, 0, sizeof(EC_T_RAS_SERVER_PARMS));
    EC_T_PVOID          pvRemApiHandle;
    oRemoteApiConfig.dwSignature            = EC_RAS_SERVER_SIGNATURE;
    oRemoteApiConfig.dwSize                 = sizeof(EC_T_RAS_SERVER_PARMS);
    /* INADDR_ANY */
    oRemoteApiConfig.oAddr.dwAddr           = 0;
    /*< default is port 6000 > */
    oRemoteApiConfig.wPort                  = wServerPort;
    /*< default is 2 ms */
    oRemoteApiConfig.dwCycleTime            = EC_RAS_CYCLE_TIME;
    oRemoteApiConfig.dwCommunicationTimeout = EC_RAS_MAX_WATCHDOG_TIMEOUT;
    EC_CPUSET_ZERO(oRemoteApiConfig.oAcceptorThreadCpuAffinityMask);
    oRemoteApiConfig.dwAcceptorThreadPrio           = MAIN_THREAD_PRIO;
    oRemoteApiConfig.dwAcceptorThreadStackSize      = JOBS_THREAD_STACKSIZE;
    EC_CPUSET_ZERO(oRemoteApiConfig.oClientWorkerThreadCpuAffinityMask);
    oRemoteApiConfig.dwClientWorkerThreadPrio       = MAIN_THREAD_PRIO;
    oRemoteApiConfig.dwClientWorkerThreadStackSize  = JOBS_THREAD_STACKSIZE;
    /* RAS notification callback function */
    oRemoteApiConfig.pfnRasNotify     = RasNotifyCallback;
    /* RAS notification callback function context */
    oRemoteApiConfig.pvRasNotifyCtxt = pAppContext->pNotificationHandler;
    /* pre-allocation */
    oRemoteApiConfig.dwMaxQueuedNotificationCnt = 100;
    /* pre-allocation */
    oRemoteApiConfig.dwMaxParallelMbxTferCnt       = 50;
    /* span between to consecutive cyclic notifications of same type */
    oRemoteApiConfig.dwCycErrInterval              = 500;

    /* init simulator */
    esInitSimulator(...);

    /* start remote API server */
    esRasSrvStart(dwSimulatorInstanceId, &oRemoteApiConfig, &pvRemApiHandle);

    /* stop remote API server */
    esRasSrvStop(dwSimulatorInstanceId, pvRemApiHandle, 2000)

    esDeinitSimulator(...);
}
```

3.8.3 Required API Calls

esRasSrvStart

EC_T_DWORD EC_NAMESPACE::**esRasSrvStart** (
    const *EC_T_RAS_SERVER_PARMS* *pParms,
    EC_T_PVOID *ppHandle
)

Initializes and starts a remote API Server Instance.

Parameters

- **pParms** – [in] Server start-up parameters
- **ppHandle** – [out] Handle to opened instance, used for ctrl access

Returns

EC_E_NOERROR or error code

struct **EC_T_RAS_SERVER_PARMS**

Public Members

EC_T_DWORD **dwSignature**
[in] Set to EC_RAS_SERVER_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(EC_T_RAS_SERVER_PARMS)

EC_T_LOG_PARMS **LogParms**
[in] Logging parameters

EC_T_IPADDR **oAddr**
[in] Remote Access Server (RAS) listen IP address

EC_T_WORD **wPort**
[in] Remote Access Server (RAS) listen port

EC_T_WORD **wMaxClientCnt**
[in] Max. clients in parallel (0: unlimited)

EC_T_DWORD **dwCycleTime**
[in] Cycle Time of RAS Network access (acceptor, worker)

EC_T_DWORD **dwCommunicationTimeout**
[in] Timeout [ms] before automatically closing connection

EC_T_CPUSET **oAcceptorThreadCpuAffinityMask**
[in] Acceptor Thread CPU affinity mask

EC_T_DWORD **dwAcceptorThreadPrio**
[in] Acceptor Thread Priority

EC_T_DWORD **dwAcceptorThreadStackSize**
[in] Acceptor Thread Stack Size

EC_T_CPUSET **oClientWorkerThreadCpuAffinityMask**
[in] Client Worker Thread CPU affinity mask

EC_T_DWORD **dwClientWorkerThreadPrio**
[in] Client Worker Thread Priority

EC_T_DWORD **dwClientWorkerThreadStackSize**
[in] Client Worker Thread Stack Size

EC_T_DWORD **dwMaxQueuedNotificationCnt**
[in] Amount of concurrently queue able Notifications

EC_T_DWORD **dwMaxParallelMbxTferCnt**
[in] Amount of concurrent active mailbox transfers

EC_PF_NOTIFY **pfnRasNotify**
[in] Function pointer called to notify error and status information generated by Remote API Layer

EC_T_VOID ***pvRasNotifyCtxt**
[in] Notification context returned while calling pfNotification

EC_T_DWORD **dwCycErrInterval**
[in] Interval which allows cyclic Notifications

EC_T_DWORD **dwMaxQueuedNotificationSize**
[in] Size of concurrent active mailbox transfers

EC_PF_CHECK_TOKEN **pfCheckToken**
[in] Function pointer called to check token

EC_T_VOID ***pvCheckTokenContext**
[in] Check token context

EC_T_BYTE ***pbyTlsCert**
[in] TLS certificate filename string

EC_T_DWORD **dwTlsCertSize**
[in] Size of TLS certeficate

EC_T_TLS_CERT_TYPE **eTlsCertType**
[in] TLS certeficate type

EC_T_BYTE ***pbyTlsPrivKey**
[in] TLS private key filename string

EC_T_DWORD **dwTlsPrivKeySize**
[in] Size of TLS private key

EC_T_TLS_PRIVKEY_TYPE **eTlsPrivKeyType**
[in] TLS certeficate type

union **EC_T_IPADDR**
#include <EthernetServices.h>

Public Members

EC_T_INNER_IPADDR **sAddr**
IPv4 address (endianness independent)

EC_T_DWORD **dwAddr**
Reserved, use EC_T_IPADDR::sAddr.by instead. OS-Layer socket API calls (SOCK-ADDR_IN::sin_addr).

struct **EC_T_INNER_IPADDR**

Public Members

EC_T_BYTE **by**[4]
IPv4 address (endianness independent)

esRasSrvStop

EC_T_DWORD EC_NAMESPACE::**esRasSrvStop**(
EC_T_PVOID pvHandle,
EC_T_DWORD dwTimeout
)
Stop and de-initialize remote API Server Instance.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **dwTimeout** – [in] Timeout [ms] used to shut down all spawned threads, it's multiplied internally by the amount of threads spawned

Returns
EC_E_NOERROR or error code

RasNotifyCallback - xxx

Callback function called by Remote API Server in case of State changes or error situations.

Data structure filled with detailed information about the according notification.

struct *EC_T_NOTIFYPARMS*

EC_T_VOID **pCallerData*

EC_T_BYTE **pbyInBuf*

EC_T_DWORD *dwInBufSize*

EC_T_BYTE **pbyOutBuf*

EC_T_DWORD *dwOutBufSize*

EC_T_DWORD **pdwNumOutData*

RasNotifyCallback - EC_RAS_NOTIFY_CONNECTION

Notification about a change in the Remote API's state.

RasNotifyCallback - EC_RAS_NOTIFY_CONNECTION

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_RAS_CONNOTIFYDESC
- `dwInBufSize`: [in] Size of the input buffer in bytes

- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Data structure containing the new Remote API state and the cause of state change.

struct **EC_T_RAS_CONNOTIFYDESC**

Public Members

EC_T_DWORD **dwCause**
[in] Cause of state connection state change

EC_T_DWORD **dwCookie**
[in] Unique identification cookie of connection instance

RasNotifyCallback - EC_RAS_NOTIFY_REGISTER

Notification that a connected application registered a client to the stack.

RasNotifyCallback - EC_RAS_NOTIFY_REGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_RAS_REGNOTIFYDESC
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_RAS_REGNOTIFYDESC**

Public Members

EC_T_DWORD **dwCookie**
[in] Unique identification cookie of connection instance

EC_T_DWORD **dwResult**
[in] Result of registration request

EC_T_DWORD **dwInstanceId**
[in] Master Instance client registered to

EC_T_DWORD **dwClientId**
[in] Client ID of registered client

RasNotifyCallback - EC_RAS_NOTIFY_UNREGISTER

Notification that a connected application un-registered a client from the simulator stack.

RasNotifyCallback - EC_RAS_NOTIFY_UNREGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_RAS_REGNOTIFYDESC
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

RasNotifyCallback - EC_RAS_NOTIFY_REGISTER

RasNotifyCallback - EC_RAS_NOTIFY_MARSHALERROR

Notification about an error during marshalling in Remote API Server connection layer.

RasNotifyCallback - EC_RAS_NOTIFY_MARSHALERROR

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_RAS_MARSHALERRORDESC
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_RAS_MARSHALERRORDESC**

Public Members

EC_T_DWORD **dwCookie**
[in] Unique identification cookie of connection instance

EC_T_DWORD **dwCause**
[in] Cause of the command marshalling error

EC_T_DWORD **dwLenStatCmd**
[in] Length of the faulty command

EC_T_DWORD **dwCommandCode**
[in] Command code of the faulty command

RasNotifyCallback - EC_RAS_NOTIFY_ACKERROR

Notification about an error during creation of ack / nack packet.

RasNotifyCallback - EC_RAS_NOTIFY_ACKERROR

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing error code
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

RasNotifyCallback - EC_RAS_NOTIFY_NONOTIFYMEMORY

Notification given, when no empty buffers for notifications are available in pre-alloced notification store. This points to a configuration error.

RasNotifyCallback - EC_RAS_NOTIFY_NONOTIFYMEMORY

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing a unique identification cookie of connection instance
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

RasNotifyCallback - EC_RAS_NOTIFY_STDNOTIFYMEMORYSMALL

Notification given when the buffer size for standard notifications available in pre-alloced notification store is too small to carry a specific notification. This points to a configuration error.

RasNotifyCallback - EC_RAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Notification given when the buffer size for Mailbox notifications available in pre-alloced notification store is too small to carry a specific notification. This points to a configuration error.

RasNotifyCallback - EC_RAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance
- `dwInBufSize`: [in] Size of the input buffer in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0

- `pdwNumOutData`: [out] Should be set to EC_NULL

This is a serious error. If this error occurs, Mailbox Transfer objects may have gotten out of sync and are therefore no longer valid.

See also:

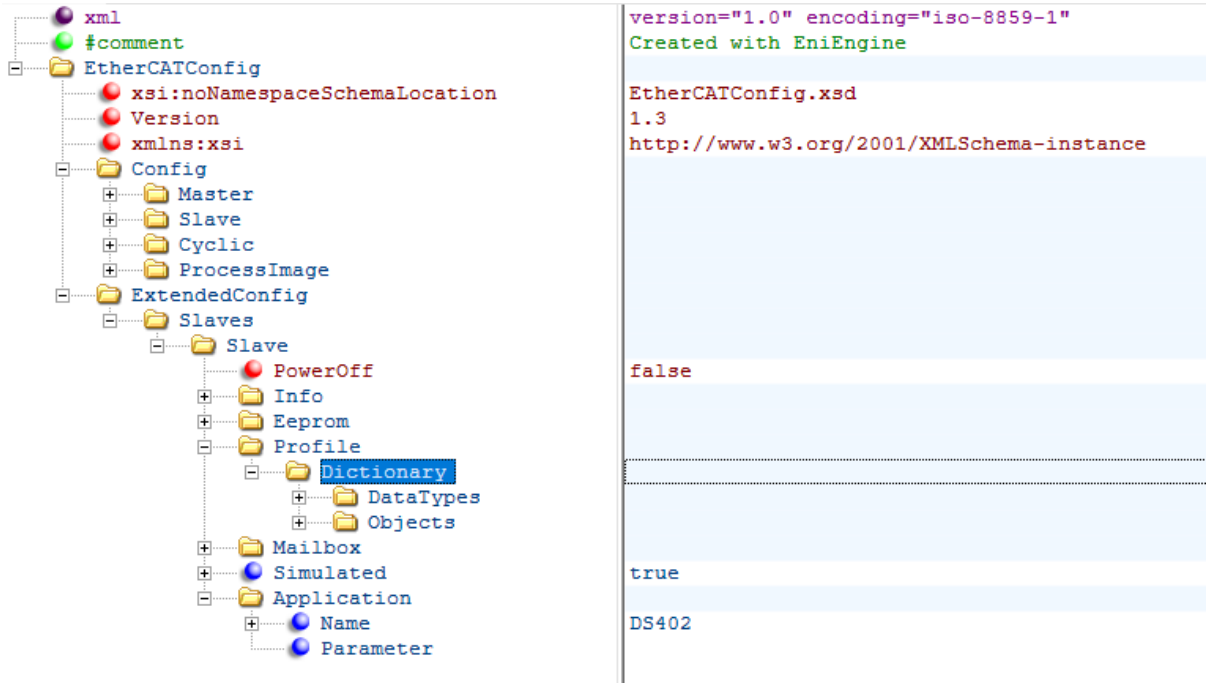
`esRasSrvStart()` for Mailbox notifications should be dimensioned correctly.

3.9 Motion

3.9.1 Internal DS402 Simulation

The EC-Simulator SiL includes an internal DS402 state machine simulation with actual position emulation. Axis movement is implemented by mapping Target position (0x607A) directly to Position actual value (0x6064) in CSP mode, and Target velocity (0x60FF) directly to Velocity actual value (0x606C) in CSV mode.

To enable the internal DS402 simulation the application name of the simulated SubDevice must be set to “DS402” in the EXI file exported by EC-Engineer:



3.9.2 EcSimulatorSilDemoMotion

For basic functionality it suffices to use the internal DS402 Simulation as demonstrated with EcSimulatorSilDemo and an EXI file with the application name of the simulated SubDevice set to “DS402”. It is also possible to implement the motion simulation in the application which needs more source code changes. The EcSimulatorSilDemoMotion is an extension of the EcSimulatorSilDemo that shows how to integrate the DS402 simulation into the EC-Simulator application. It handles the following additional tasks:

- Simulating CiA402 capabilities of SubDevices

The EcSimulatorSilDemoMotion is available “out of the box” for different operating systems.

See also:

EcSimulatorSilDemo

There is also support for multiple axes.

3.9.3 Command line parameters

EcSimulatorSilDemoMotion supports all parameters supported by *EcSimulatorSilDemo* .

The following additional parameters are supported:

- "-ds402 <FixedAddress1,…,FixedAddressN>": Simulate CiA402 capability of SubDevices. SubDevices are passed as comma separated list of fixed SubDevice addresses to simulate.

The Demo uses the following files in addition to the files used by *EcSimulatorSilDemo*:

- `Examples/Common/EcSimulatorDs402.h`: Simulation of CiA402 capabilities using SubDevice application's callback functions, see `esSetSlaveSscApplication()` .

4 Platform and Operating Systems (OS)

4.1 tenAsys INtime

Real-time Ethernet Drivers are available for INtime. If using INtime with Windows running in parallel on the same host the network adapter has to be assigned to INtime. The network adapters should be passed to INtime using the `INtime Device Manager`. Please refer to the INtime user manual for this.

Search locations for Real-time Ethernet Drivers can be adjusted using the `PATH` environment variable.

4.1.1 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for INtime.

Extra include paths

```
<InstallPath>\SDK\INC\INtime
<InstallPath>\Examples\Common\INtime
```

Extra source paths

```
<InstallPath>\Examples\Common\INtime
<InstallPath>\Sources\OsLayer\INtime
```

Extra library paths to the main EtherCAT® components

```
<InstallPath>\SDK\LIB\INtime
```

4.2 Linux

4.2.1 OS optimizations

Linux itself is not real-time capable, so it is recommended to use it with the additional *PREEMPT_RT* patch.

The power management can disrupt cyclical processing, it is advisable to disable the *CPUIDLE sub-system* and *CPUFREQ sub-system*. The sub-systems can be disabled by changing the kernel command line parameters in the boot loader. On x86, x86_64 systems this is usually *GRUB*, on embedded devices with ARM, ARM64 is usually *u-boot*. It is also possible to build a custom kernel without these sub-systems.

Running an EC-Simulator application on a dedicated CPU core that is isolated from the Linux scheduler (*ISOLCPUS*) can provide additional stability.

CPUIDLE sub-system

Check if CPUFREQ sub-system is enabled:

```
$ ls /sys/devices/system/cpu/
```

If `cpuidle` appears in the list, it is enabled.

Disable CPUIDLE via the kernel command-line in GRUB:

```
linux  /boot/vmlinuz-4.19.0-16-rt-amd64 cpuidle.off=1
```

CPUFREQ sub-system

Check if CPUFREQ sub-system is enabled:

```
$ ls /sys/devices/system/cpu/
```

If `cpufreq` appears in the list, it is enabled.

Disable CPUFREQ sub-system via the kernel command-line GRUB:

```
linux  /boot/vmlinuz-4.19.0-16-rt-amd64 cpufreq.off=1
```

If CPUFREQ is not to be deactivated, the governor should be set to `performance`.

The currently active governor can be determined as follows:

```
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

The available governors with:

```
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_available_governors
```

To change governor use:

```
$ echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

ISOLCPUS

Isolate CPU core number 4 of a quad-core processor via the kernel command-line GRUB:

```
linux  /boot/vmlinuz-4.19.0-16-rt-amd64 isolcpus=3
```

`isolcpus` alone removes scheduler tasks from selected CPUs, but does not prevent timer interrupts, RCU callbacks, or network adapter IRQs. To fully isolate a CPU for real-time workloads, `nohz_full`, `rcu_nocbs`, and `irqaffinity` should be used together to eliminate kernel noise and ensure deterministic execution.

Enhanced isolation of CPU core 4 via the kernel command-line GRUB:

```
linux  /boot/vmlinuz-4.19.0-16-rt-amd64 isolcpus=3 nohz_full=3 rcu_nocbs=3 rcu_
↪nocb_poll irqaffinity=0-2
```

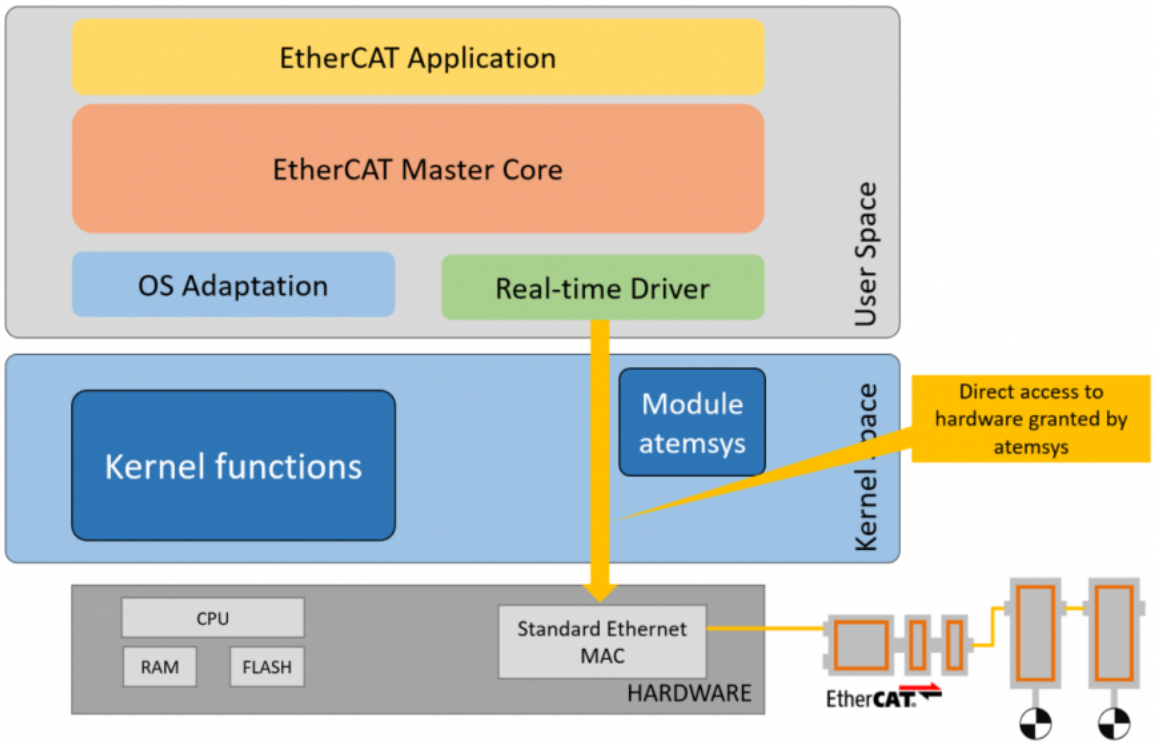
Running EcSimulatorHilDemo on the isolated CPU core by setting the CPU affinity `-a`:

```
$ ./EcSimulatorHilDemo -a 3
```

4.2.2 atemsys kernel module

To use Real-time Ethernet Drivers under Linux, the atemsys kernel module must be compiled and loaded. atemsys grants direct access to hardware to improve the performance.

All necessary scripts, source code and a detailed description of the installation can be found on https://github.com/acontis/atemsys. A ready-to-use Yocto recipe is also available on https://github.com/acontis/meta-acontis



atemsys as Device Tree Ethernet Driver

atemsys can also be used as a device tree driver to avoid certain conflicts between the Real-time Ethernet Driver and the Linux kernel, e.g. power management, shared MDIO bus, etc..

A detailed guide on how to customize the device tree accordingly can also be found on https://github.com/acontis/atemsys. Example device tree modifications for different Real-time Ethernet Drivers/SoC can be found in https://github.com/acontis/atemsys/wiki.

Note: This is the preferred solution on all embedded devices with device tree support.

atemsys and PHY OS Driver

To use the PHY OS Driver, the acontis kernel module atemsys has to be included in the kernel device tree as an official driver for the Ethernet controller and doesn't require any additional configuration at the application level. As a result atemsys can interact with Linux drivers.

4.2.3 Unbind Ethernet Driver instance

Ethernet Driver instances used by Real-time Ethernet Drivers may not be bound by kernel driver modules! Unbind can be done by unloading the kernel driver module, via the unbind interface of the driver or by modifying the device tree.

Unbind from kernel driver

The following command unbinds an instance without unloading the kernel driver module:

PCI

```
$ echo "<Instance-ID>" > /sys/bus/pci/drivers/<driver-name>/unbind
```

Example:

```
$ echo "0000:00:19.0" > /sys/bus/pci/drivers/e1000e/unbind
```

This call requires the PCI bus, device, function codes (in the above example it is 0000:00:19.0). The codes can be found using Linux commands like, for example:

```
$ ls /sys/bus/pci/drivers/e1000e
```

SoC

```
$ echo "<Instance-ID>" > /sys/bus/platform/drivers/<driver-name>/unbind
```

Example:

```
$ echo "2188000.ethernet" > /sys/bus/platform/drivers/fec/unbind
```

Unload kernel driver

Not all drivers allow unbinding of network interfaces. If unbinding is not supported the corresponding Linux kernel driver must not be loaded.

The following command lists the loaded kernel modules that may conflict with Real-time Ethernet Driver:

```
$ lsmod | egrep "<module-name>"
```

Example:

```
$ lsmod | egrep "e1000|e1000e|igb"
```

PCI/PCIe: The command *lspci -v* shows which driver is assigned to which network card, e.g.:

```
$ lspci -v
```

```
...
11:0a.0 Ethernet controller: Intel Corporation 82541PI Gigabit Ethernet Controller␣
↪(rev 05)
...
Kernel driver in use: e1000e
```

Modules can be prevented from loading with the following commands:

```
$ echo blacklist <module-name> | sudo tee -a /etc/modprobe.d/blacklist.conf
$ update-initramfs -k all -u
$ sudo reboot
```

The following table shows the Kernel modules related to the Real-time Ethernet Driver:

Chip	Real-time Ethernet Driver	Kernel driver(s)	Remarks
Broadcom Genet	emllBcmGenet	genet	Unbind not supported
Beckhoff CCAT	emllCCAT	ec_bhf	
CPSW	emllCPSW	ti_cpsw	
Generic	emllDpdk		
DesignWare 3504	emllDW3504	stmmac	
	emllEG20T		
Freescale TSEC/eTSEC v1/2	emllETSEC	gianfar_driver	
Freescale FEC and ENET controller	emllFslFec	fec, fec_ptp	
Cadence GEM/MACB	emllGEM	gem, macb	
Intel Pro/1000	emllI8254x	igb, e1000, e1000e	
Intel Pro/1000	emllIntelGbe	igb, e1000, e1000e	
Intel Pro/100	emllI8255x	e100	
ICSS	emllICSS	prueth,pruss	Unbind not supported
RDC R6040	emllR6040		
Realtek RTL8139	emllRTL8139	8139too, 8139cp	
Realtek RTL8169 / RTL8111 / RTL8168	emllRTL8169	r8169	Unbind not supported
SuperH	emllSHEth	sh_eth	Unbind not supported
Generic	emllSockRaw		
Generic	emllSockXdp		

4.2.4 Docker

It is possible to operate EC-Simulator within a Docker container with realtime priority. The atemsys kernel module should be installed on the host in order to operate the container with the lowest possible capabilities and privileges.

The following additional settings, permissions for `docker run` are required:

Add atemsys device to container

```
--device=/dev/atemsys:/dev/atemsys
```

Allow max realtime priority

```
--ulimit rtprio=99
```

Add capability to set priority an lock memory

```
--cap-add=sys_nice
--cap-add=ipc_lock
```

Publish RAS server port *6000*

```
-p 6000:6000
```

4.2.5 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for Linux

Possible ARCHs (see ATECAT_ARCHSTR in `SDK/INC/Linux/EcOsPlatform.h`):

- aarch64 (ARM 64Bit)
- armv7-vfp-eabihf (ARM 32Bit)
- PPC (PPC 32Bit with "-te500v2")
- riscv64 (RISC-V 64Bit)
- x64 (x86 64Bit)
- x86 (x86 32Bit)

An ARM VFP system returns success on

```
$ readelf -A /proc/self/exe | grep Tag_ABI_VFP_args
```

Extra include paths

```
<InstallPath>/Examples/Common/Linux
<InstallPath>/SDK/INC/Linux
```

Extra source paths

```
<InstallPath>/Examples/Common/Linux
<InstallPath>/Sources/OsLayer/Linux
```

Extra library paths to the main EtherCAT® components

```
<InstallPath>/SDK/LIB/Linux/<Arch>
```

Extra libraries (in this order)

```
EcSimulatorRasServer EcSimulator pthread dl rt
```

4.2.6 Build using cmake on Linux

Example usage to build Linux x64 Debug with cmake:

```
$ cmake -DEC_OS=Linux -DEC_ARCH=x64
$ cmake --build .
```

4.2.7 Cross-platform development under Windows

The following steps describe how to develop Linux cross-platform developing on Windows for Linux:

```
-DEC_OS=Windows -DEC_ARCH=x64
```

1. **Install MinGW**

   - Download the latest version of MinGW from the MinGW official website https://osdn.net/projects/mingw/
   - Install the mingw-get-setup.exe tool to C:\MinGW
   - Select the "Basic Setup"
   - Apply changes

2. **Install a cross platform toolchain**

   - Download a cross-platform toolchain from e.g. the Linaro release storage server https://releases.linaro.org/components/toolchain/gcc-linaro/
   - Unpack it to C:\MinGW\opt

3. **Build using cmake on Linux**

   Example usage to build for Linux x64 Debug on Windows with cmake and ninja:

   ```
   $ cmake -DEC_OS=Linux -DEC_ARCH=x64 -DCMAKE_BUILD_TYPE=Debug .
   $ cmake --build .
   ```

4. **Build for LxWin using cmake on Linux**

   Example usage to build EcMasterDemo for Linux x64 Debug on Windows with cmake and ninja:

   ```
   Workspace/LxWin/cmake/x64/Debug> cmake.exe -G Ninja ../../../../..
   ↪  -DCMAKE_TOOLCHAIN_FILE=../../../../Linux/Toolchain.cmake -DEC_OS=LxWin
   ↪  -DEC_ARCH=x64 -DCMAKE_BUILD_TYPE=Debug
   Workspace/LxWin/cmake/x64/Debug> ninja.exe EcMasterDemo
   ```

5. **Cross build using cmake for Linux on Windows**

   Example usage to build EcMasterDemo for Linux x64 Debug on Windows with cmake and ninja:

   ```
   Workspace/Linux/cmake/x64/Debug> cmake.exe -G Ninja ../../../../..
   ↪  -DCMAKE_TOOLCHAIN_FILE=../../../Toolchain.cmake -DEC_OS=Linux -DEC_ARCH=x64
   ↪  -DCMAKE_BUILD_TYPE=Debug
   Workspace/Linux/cmake/x64/Debug> ninja.exe EcMasterDemo
   ```

6. **Cross build using Eclipse CDT on Windows**

   - Download and install the latest version of Eclipse CDT from the Eclipse official website https://projects.eclipse.org/projects/tools.cdt
   - Create a start batch file for eclipse

```
set PATH=C:\MinGW\bin;C:\MinGW\msys\1.0\bin;%LINUX_CROSS_GCC_ARM_PATH%;
↪%PATH%
set LINUX_CROSS_GCC_ARM_PATH=C:\MinGW\opt\gcc-linaro-7.3.1-2018.05-i686-
↪mingw32_aarch64-linux-gnu\bin
set CFLAGS=-IC:\MinGW\opt\gcc-linaro-7.3.1-2018.05-i686-mingw32_aarch64-
↪linux-gnu\aarch64-linux-gnu\libc\usr\include
set CROSS_COMPILE=aarch64-linux-gnu-
set ARCH=aarch64
eclipse.exe
```

4.3 QNX Neutrino

4.3.1 Thread priority

QNX supports a total of 256 scheduling priority levels. A non-root thread can set its priority to a level from 1 to 63 (the highest priority).

Using priorities higher than 63 is only possible if the allowed priority range is changed for non-root processes:

```
$ procnto -P priority
```

For more information on changing the priority range refer to the QNX documentation.

Attention: Not changing the priority range leads to poor timing performance!

4.3.2 Unbind Ethernet Driver instance

The network interface must be unloaded if it is used by an operating system driver. Depending on the QNX version, a corresponding command must be executed in the QNX Shell or the QNX Build Script.

QNX >= 6.5

```
ifconfig en1 destroy
```

QNX >= 7.1

```
umount /dev/io-sock/devs-em.so/em1
```

4.3.3 IOMMU/SMMU support

For systems that have to use an IOMMU/SMMU for security reasons, it is possible to create a predefined typed memory region that is used by the Real-time Ethernet Driver. The definition has to be done in the QNX BSP build file and the name must match following pattern:

smm_ *LinkLayerName* **-** *InstanceNumber(32Bit Hex)*

Example: Real-time Ethernet Driver emllIntelGbe with instance number 1

```
smm_emllIntelGbe-0x00000001
```

A separate typed memory region must be defined for each Real-time Ethernet Driver instance. The typed memory is automatically used by the Real-time Ethernet Driver if it matches the pattern, otherwise the default memory is used.

4.3.4 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for QNX Neutrino.

Extra include paths

```
<InstallPath>/SDK/INC/QNX
<InstallPath>/Examples/Common/QNX
```

Extra source paths

```
<InstallPath>/Examples/Common/QNX
<InstallPath>/Sources/OsLayer/QNX
```

Extra library paths to the main EtherCAT® components

```
<InstallPath>/SDK/LIB/QNX/<Arch>
```

Extra libraries (in this order)

```
EcSimulatorRasServer EcSimulator socket
```

4.4 IntervalZero RTX

EC-Simulator HiL is available for RTX64 3.x and RTX64 4.x .

4.4.1 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for RTX.

Extra include paths

```
<InstallPath>\SDK\INC\RTX
<InstallPath>\Examples\Common\RTX
```

Extra source paths

```
<InstallPath>\Examples\Common\RTX
<InstallPath>\Sources\OsLayer\RTX
```

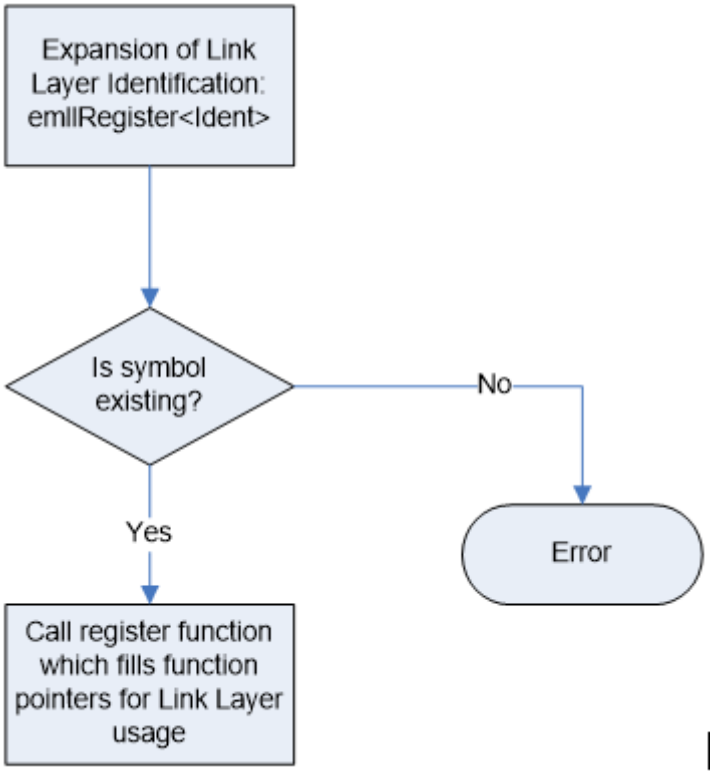
Extra library paths to the main EtherCAT® components

```
<InstallPath>\SDK\LIB\RTX64 (RTX64 4.x)
<InstallPath>\SDK\LIB\RTX64_30 (RTX64 3.x)
```

4.5 Windriver VxWorks

Real-time Ethernet Drivers for VxWorks are available. If none of the Real-time Ethernet Drivers can be used, the SNARF Ethernet Driver must be selected.

The identification of the Real-time Ethernet Driver is done like this:



4.5.1 VxWorks native

The BSP has to be prepared to support Real-time Ethernet Driver:

1. To use a Real-time Ethernet Driver the adapter memory has to be mapped into VxWorks memory space (VxWorks 5.x only). I.e. for the Intel Pro/100 Ethernet Driver this can be achieved by setting the INCLUDE_FEI_END macro in the BSP configuration file config.h.
2. To avoid conflicts with the VxWorks network driver which normally will be loaded when INCLUDE_FEI_END is set the file configNet.h has to be adjusted in a way that the network driver is not loaded. The network driver entry has to be removed from the endDevTbl[]:

```
END_TBL_ENTRY endDevTbl [] =
    {
    :       :       :
    :       :       :
    :       :       :
/*
#ifdef INCLUDE_FEI_END
    {0, FEI82557_LOAD_FUNC, FEI82557_LOAD_STRING, FEI82557_BUFF_LOAN,
    NULL, FALSE},
#endif /* INCLUDE_FEI_END */
*/
    :       :       :
    :       :       :
```

Warning: Do not call `muxDevUnload()` for a network adapter managed by a VxBus driver. VxBus drivers expect to call `muxDevUnload()` themselves in their {vxbDrvUnlink}() methods, and instability may result if `muxDevUnload()` is called for a VxBus network device instance by other code.

See also:

The VxWorks Device Driver Developer's Guide for more information about unloading VxBus devices

4.5.2 SNARF Ethernet Driver

The SNARF Ethernet Driver is only needed if none of the Real-time Ethernet Driver can be used. The appropriate network adapter drivers have to be added to the VxWorks image.

4.5.3 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for VxWorks.

Extra include paths

```
<InstallPath>/SDK/INC/VxWorks
<InstallPath>/Examples/Common/VxWorks
```

Extra source paths

```
<InstallPath>/Examples/Common/VxWorks
<InstallPath>/Sources/OsLayer/VxWorks
```

Extra library paths to the main EtherCAT® components

```
<InstallPath>/SDK/LIB/VxWorks/<ARCH>
```

GNU/PowerPC

`-mlongcall` compiler option may be needed to avoid relocation offset errors when downloading `.out` files.

4.6 Microsoft Windows

4.6.1 OS Compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for Windows.

Extra include paths

```
<InstallPath>\SDK\INC\Windows
<InstallPath>\Examples\Common\Windows
```

Extra source paths

```
<InstallPath>\Examples\Common\Windows
<InstallPath>\Sources\OsLayer\Windows
```

Extra library paths to the main EtherCAT® components

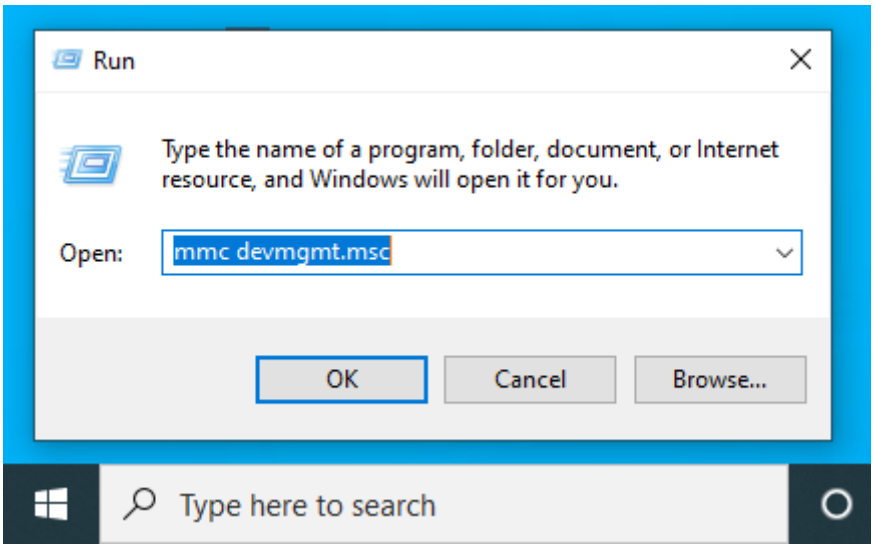
```
<InstallPath>\SDK\LIB\Windows
```

4.6.2 RtaccDevice for Real-time Ethernet Driver

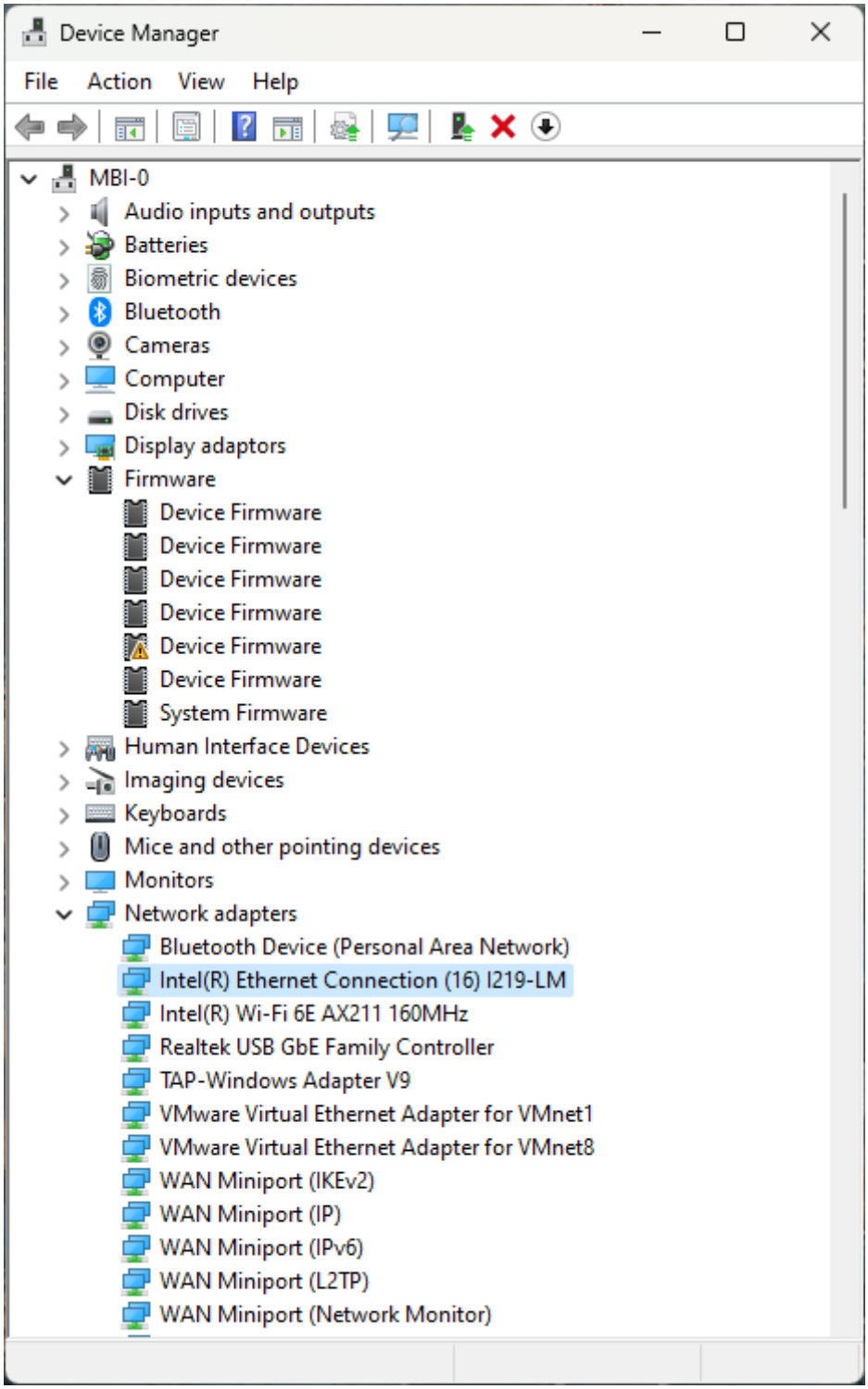
As alternative to the NDIS based or Pcap based Ethernet Driver, an optional Real-time Ethernet Driver on Windows can be installed. The Real-time Ethernet Driver replaces the original Windows driver and also requires an extra license.

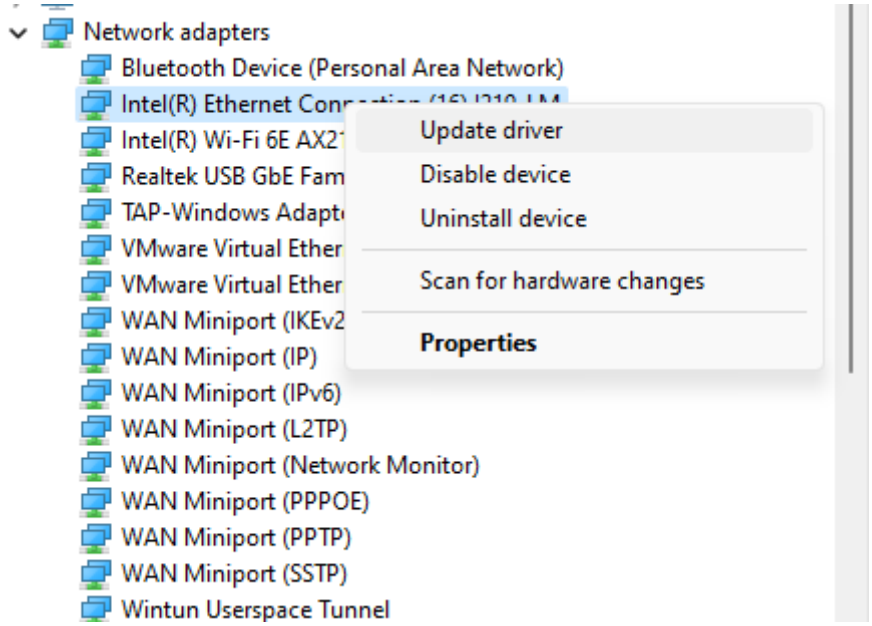
To **use the Real-time Ethernet Driver under Windows**, it is necessary to install the RtaccDevice driver included in the Real-time Ethernet Driver delivery:

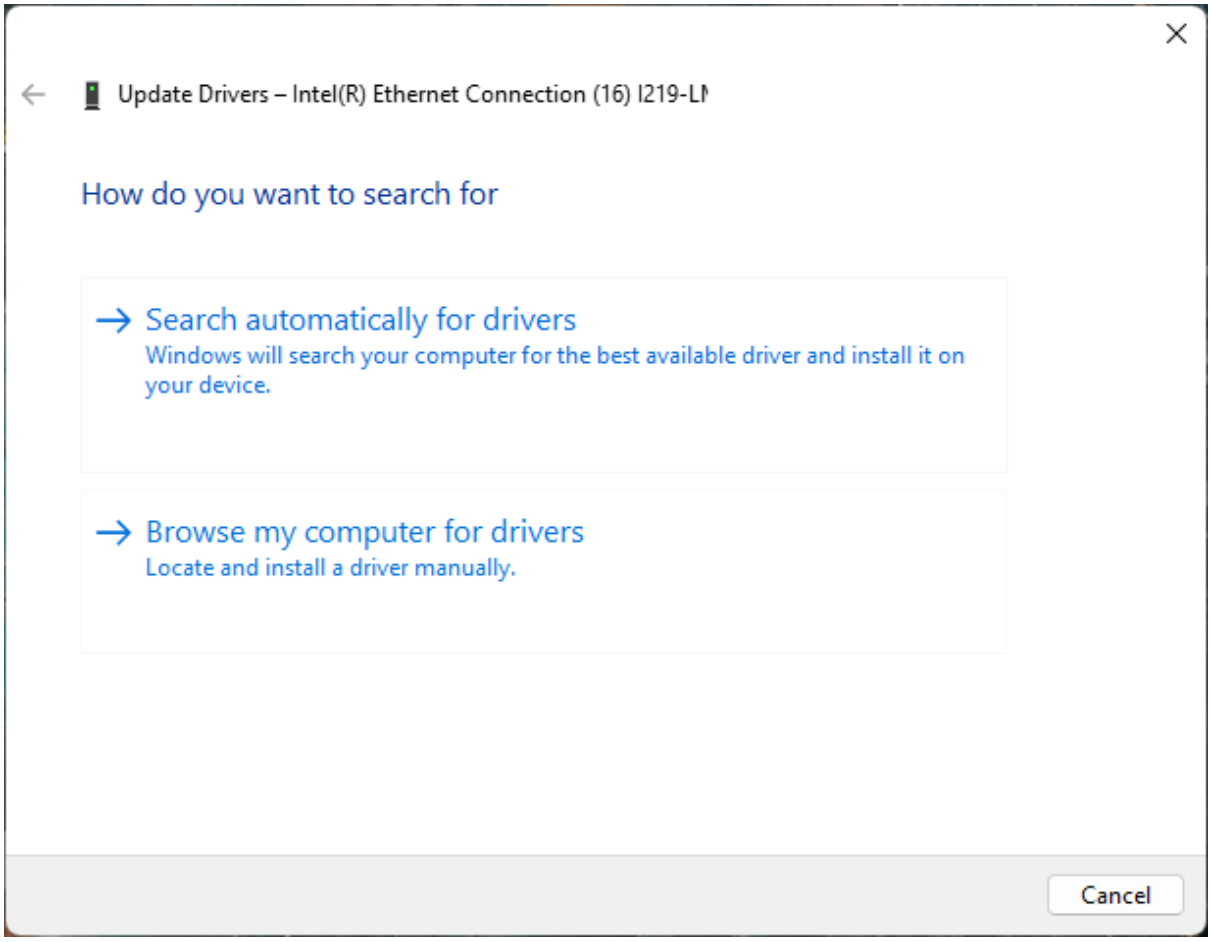
1. **Start the "Device Manager"**



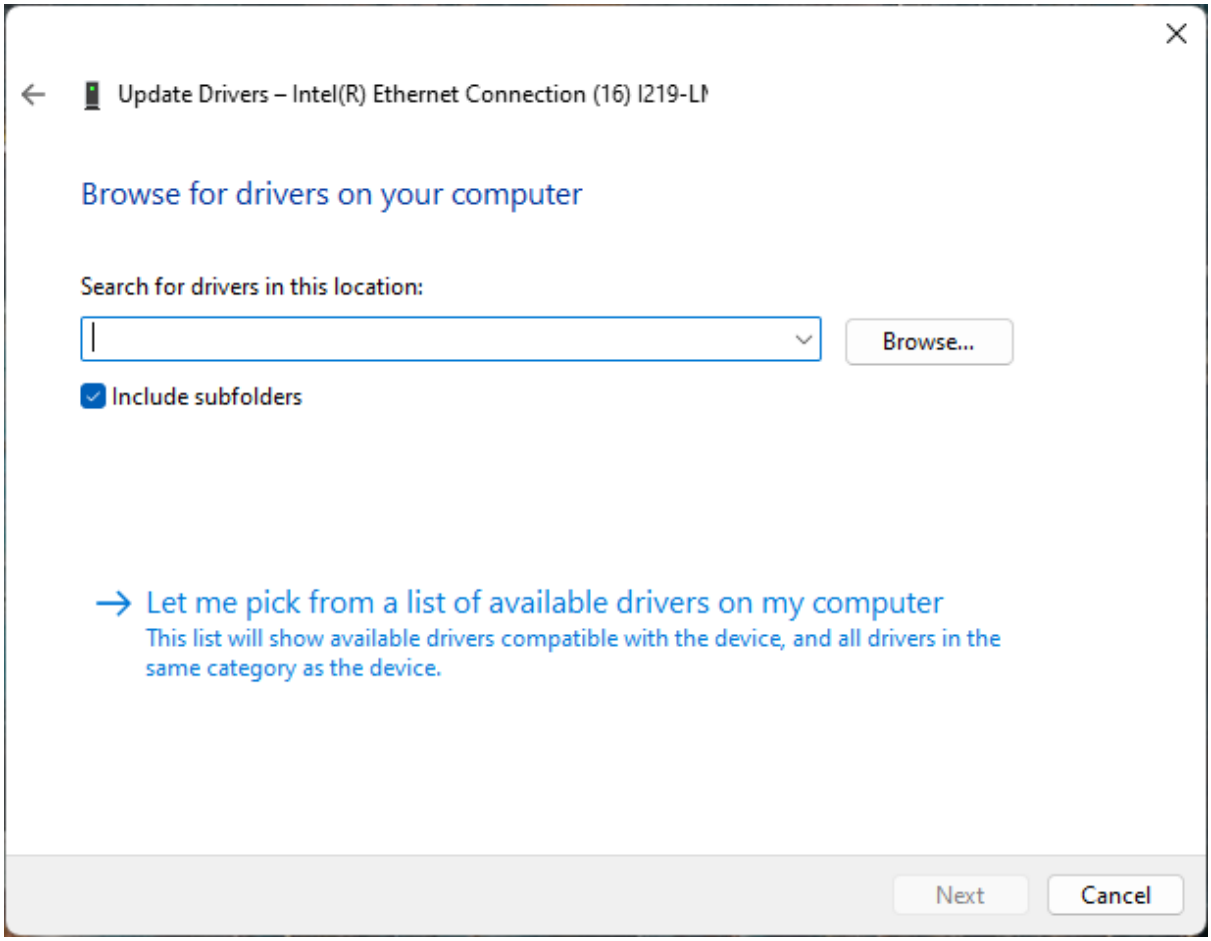
2. **Assign RtaccDevice to the network adapter**


Device Manager
File
Action
View
Help
MBI-0
Audio inputs and outputs
Batteries
Biometric devices
Bluetooth
Cameras
Computer
Disk drives
Display adaptors
Firmware
Device Firmware
Device Firmware
Device Firmware
Device Firmware
Device Firmware
Device Firmware
System Firmware
Human Interface Devices
Imaging devices
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Bluetooth Device (Personal Area Network)
Intel(R) Ethernet Connection (16) I219-LM
Intel(R) Wi-Fi 6E AX211 160MHz
Realtek USB GbE Family Controller
TAP-Windows Adapter V9
VMware Virtual Ethernet Adapter for VMnet1
VMware Virtual Ethernet Adapter for VMnet8
WAN Miniport (IKEv2)
WAN Miniport (IP)
WAN Miniport (IPv6)
WAN Miniport (L2TP)
WAN Miniport (Network Monitor)

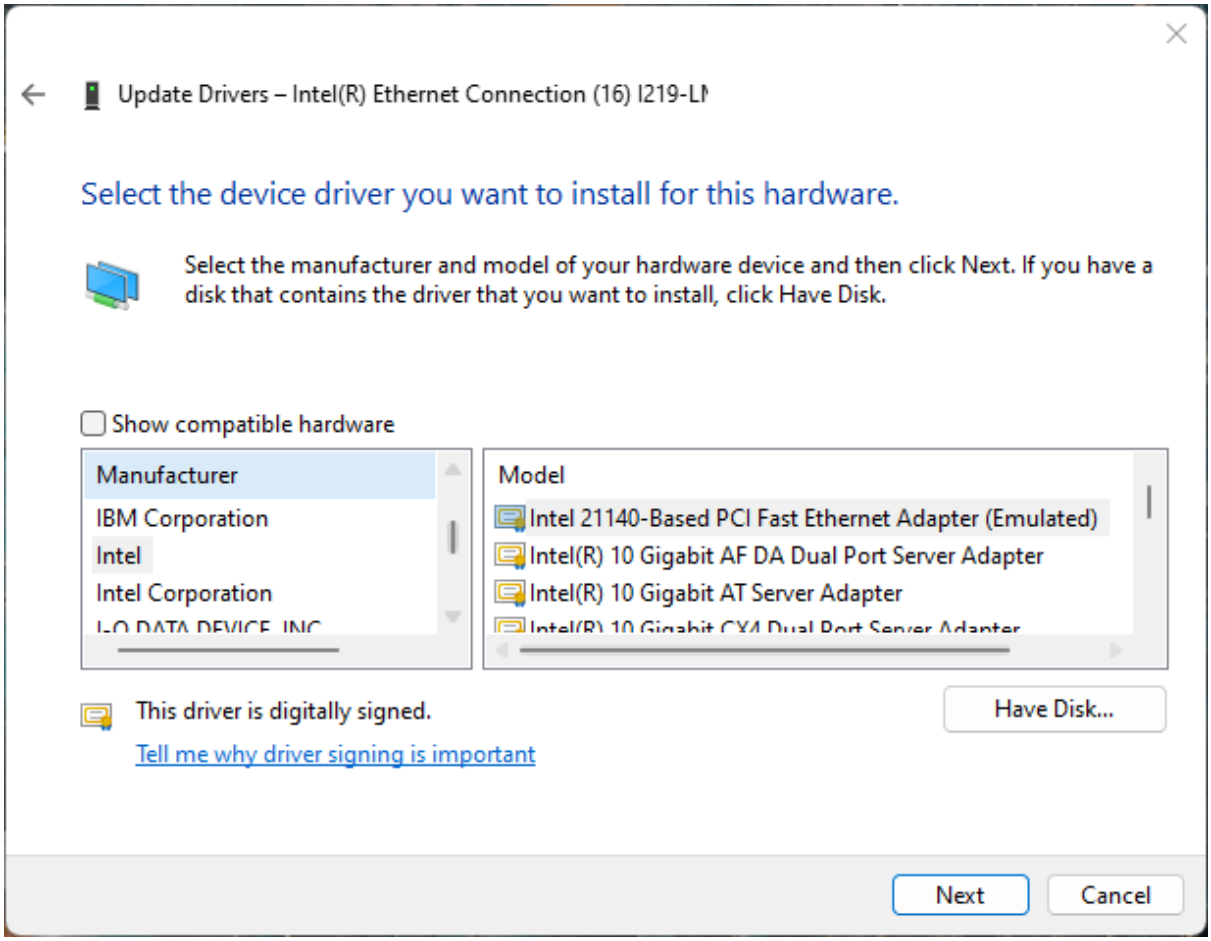




Click on “Browse my computer for drivers”



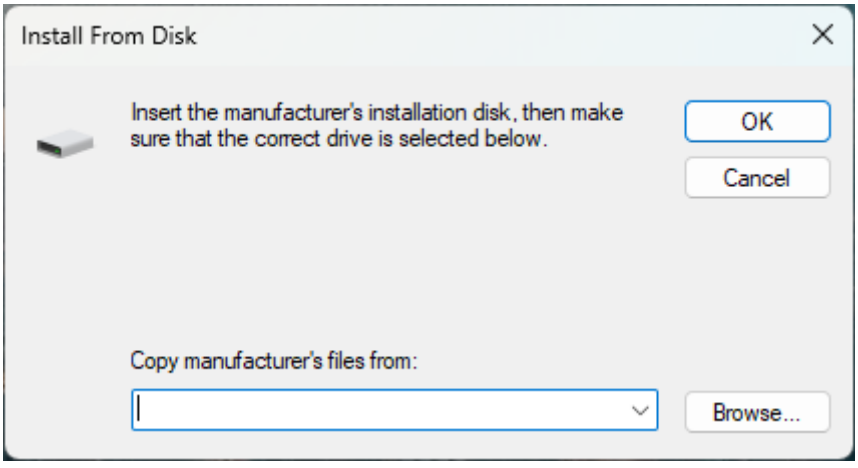
Click on “Let me pick…”



Click on “Have Disk…”

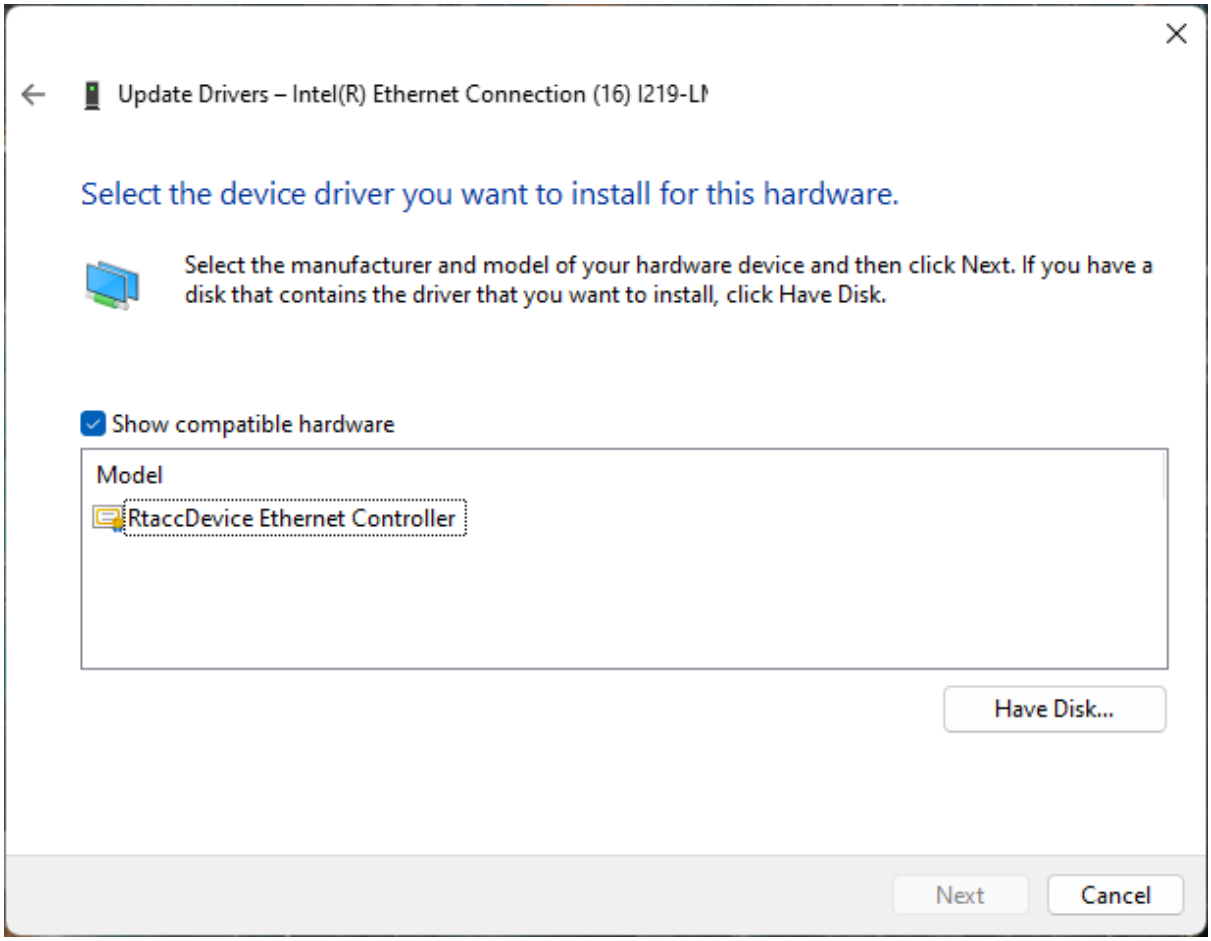
3. Enter the directory of RtaccDevice Driver

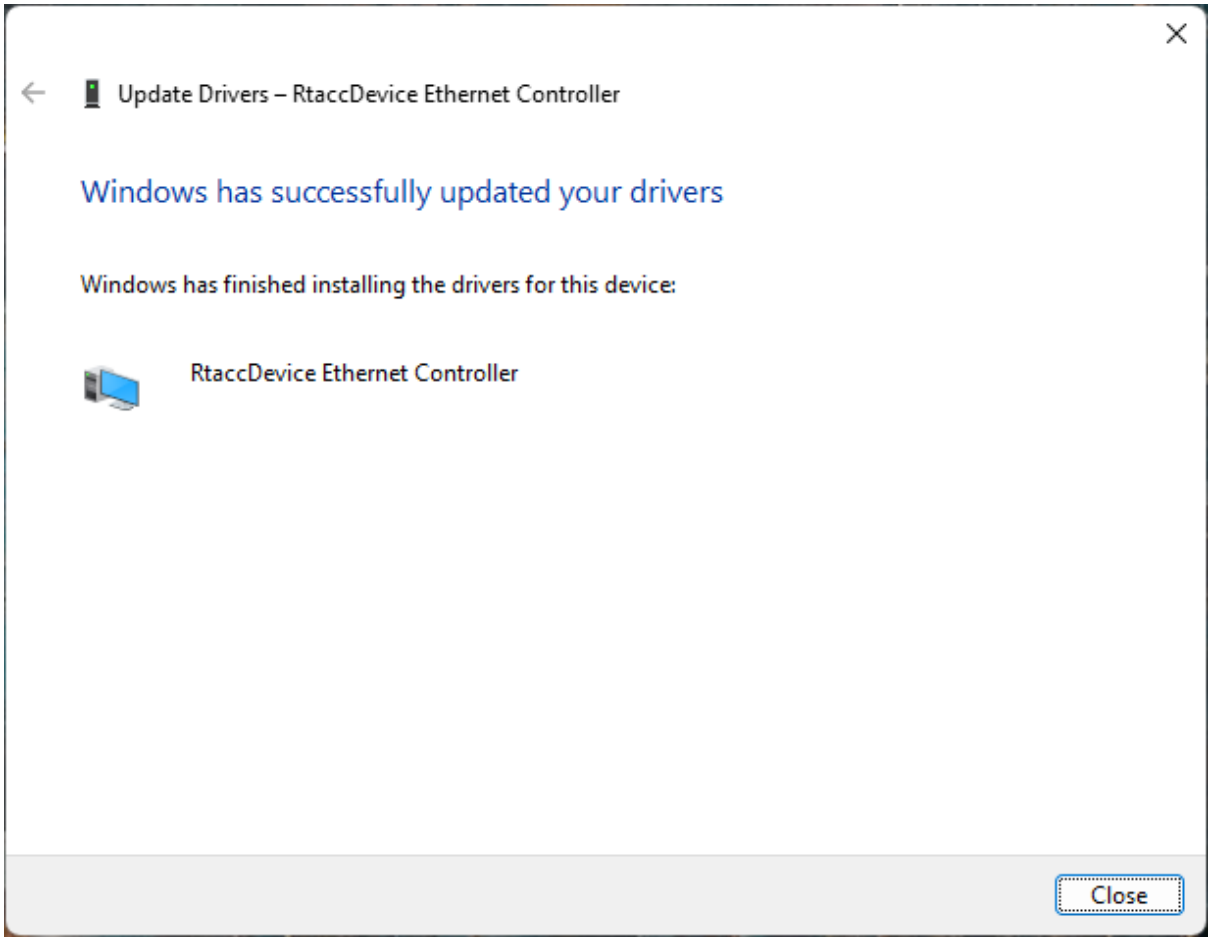
   The default folder if not changed is `<InstallPath>\Bin\Windows\x64`



   Enter the correct directory at the input box and press OK to proceed.

4. **Choose the RtaccDevice Driver and click “Next” and confirm the installation**


Update Drivers – Intel(R) Ethernet Connection (16) I219-LM
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver that you want to install, click Have Disk.
Show compatible hardware
Model
RtaccDevice Ethernet Controller
Have Disk...
Next
Cancel

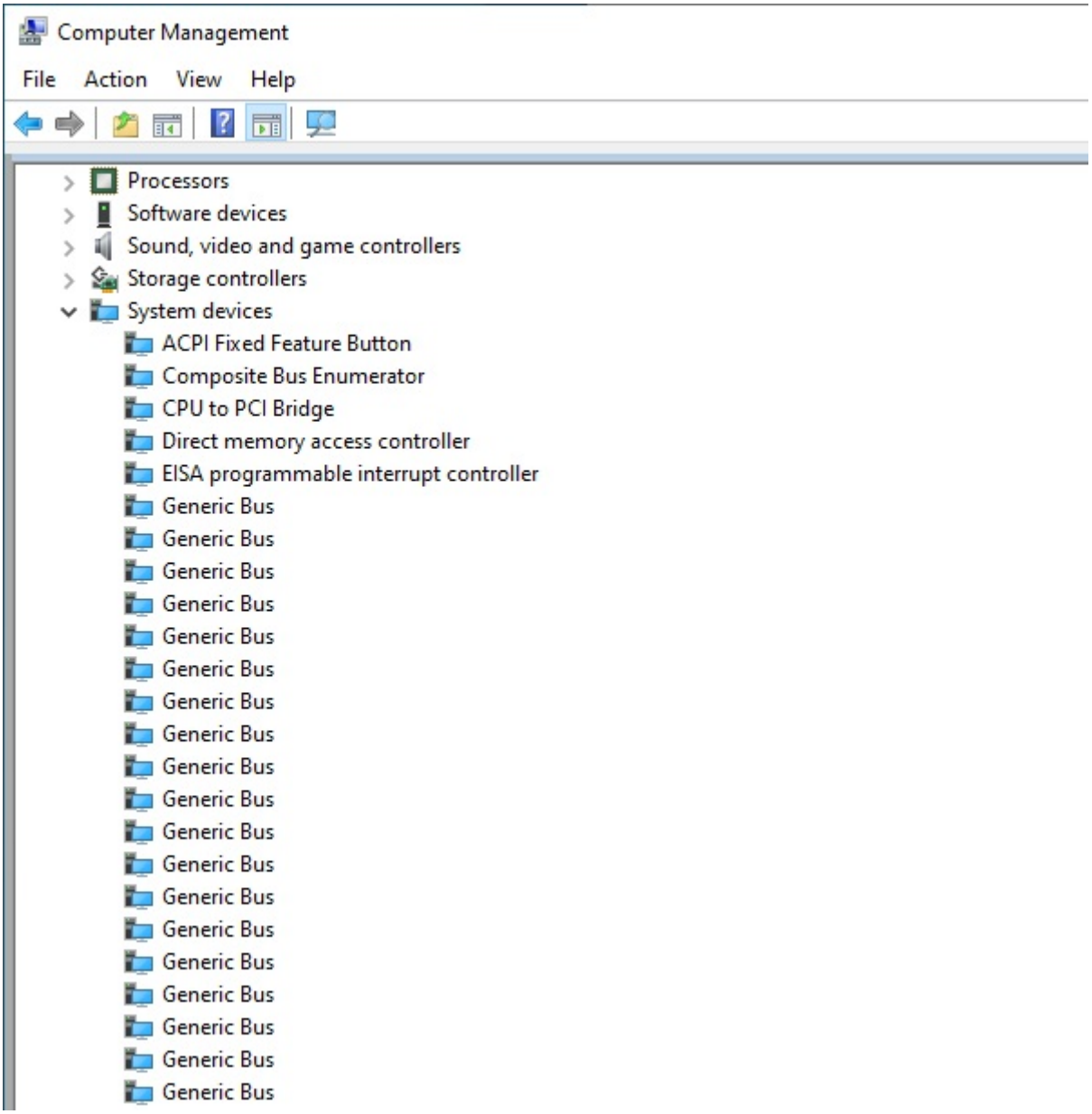


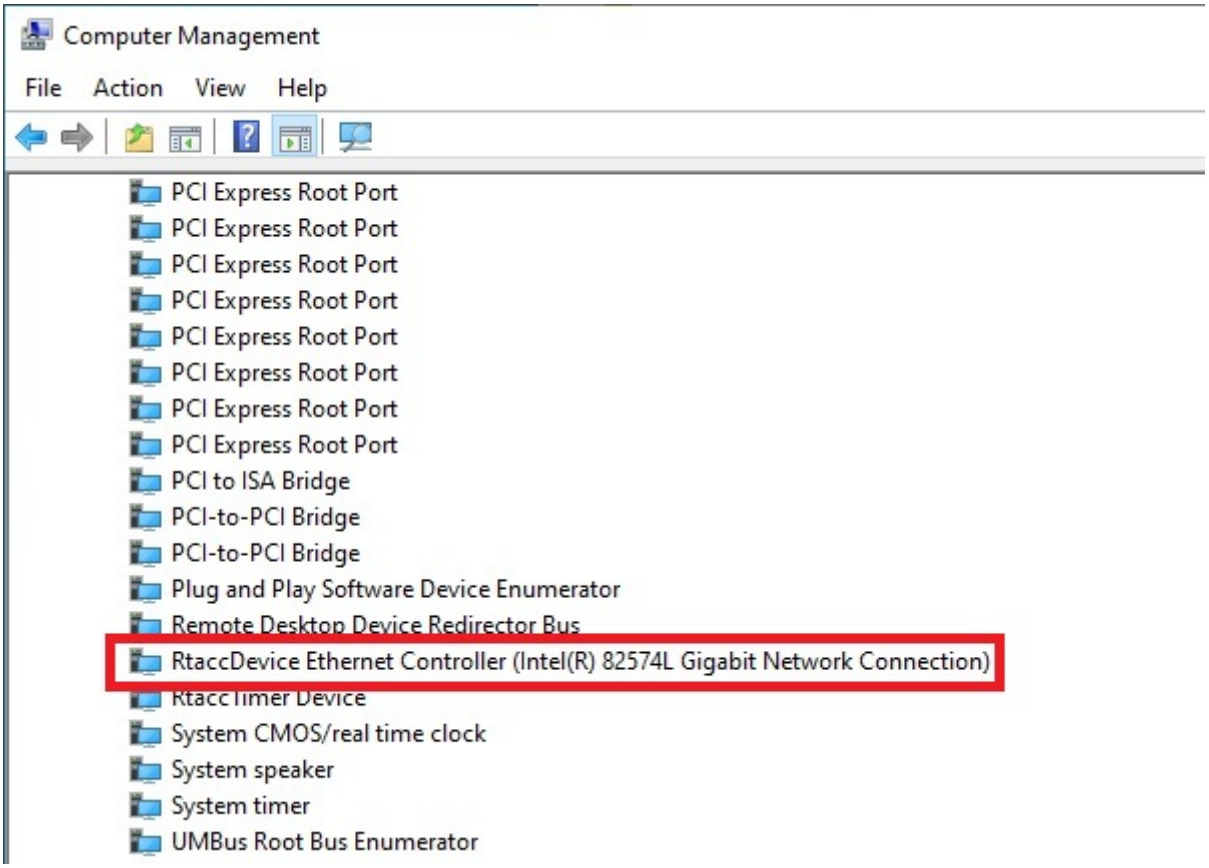
Optionally modify the search location of the Real-time Ethernet Driver

Search locations for Real-time Ethernet Driver can be adjusted using the PATH environment variable.

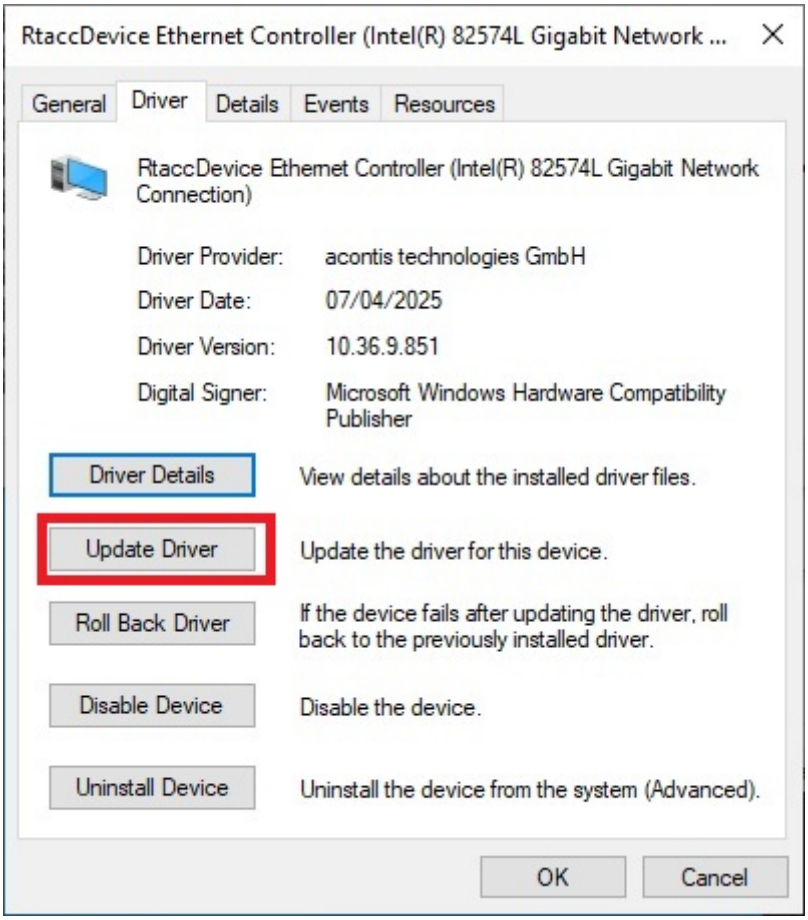
The RtaccDevice must be assigned to the network adapter for Real-time Ethernet Driver usage on Windows. In this case it is exclusively bound for EtherCAT® usage. The assignment can be reverted using the following steps:

1. Open the Windows device manager (Computer Management), then go to System devices. The RtaccDevice should be listed there:

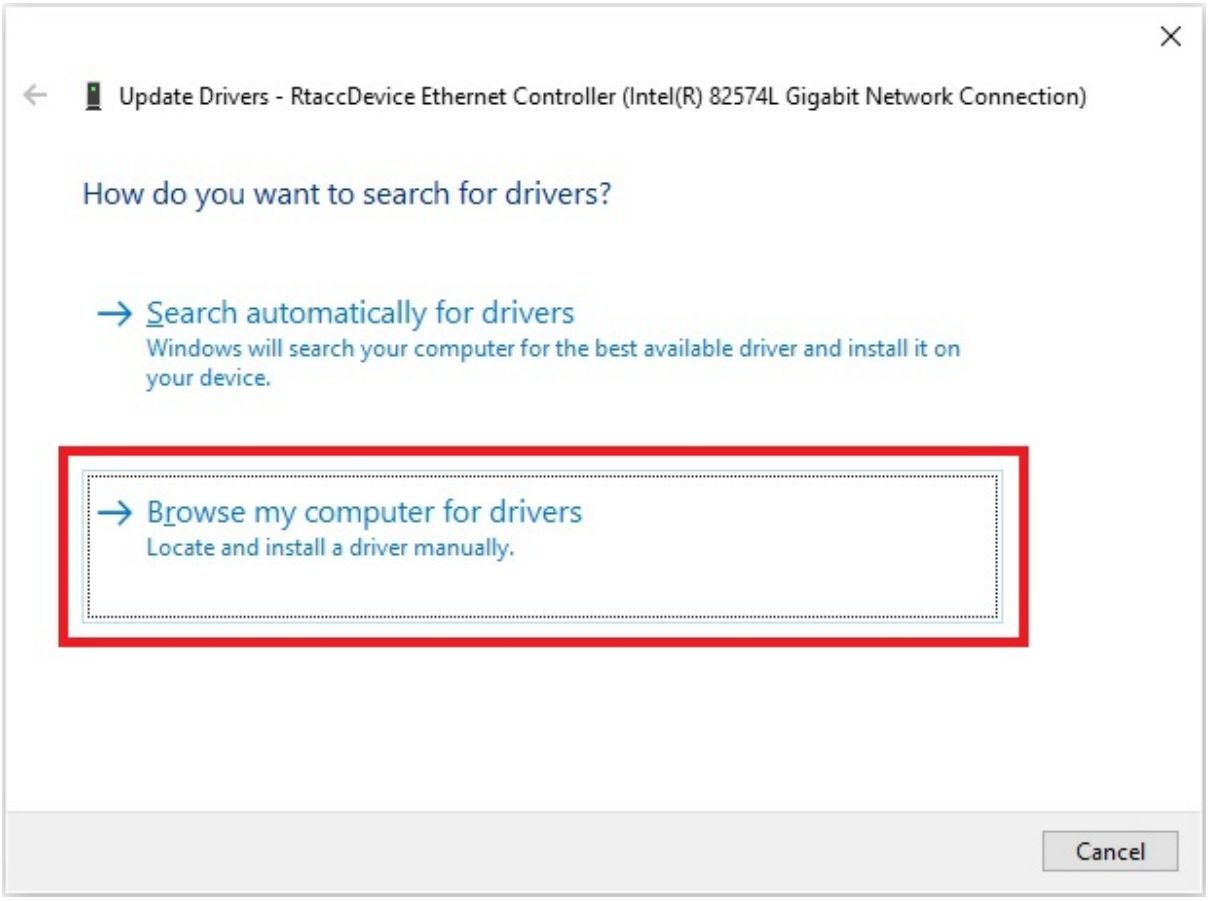




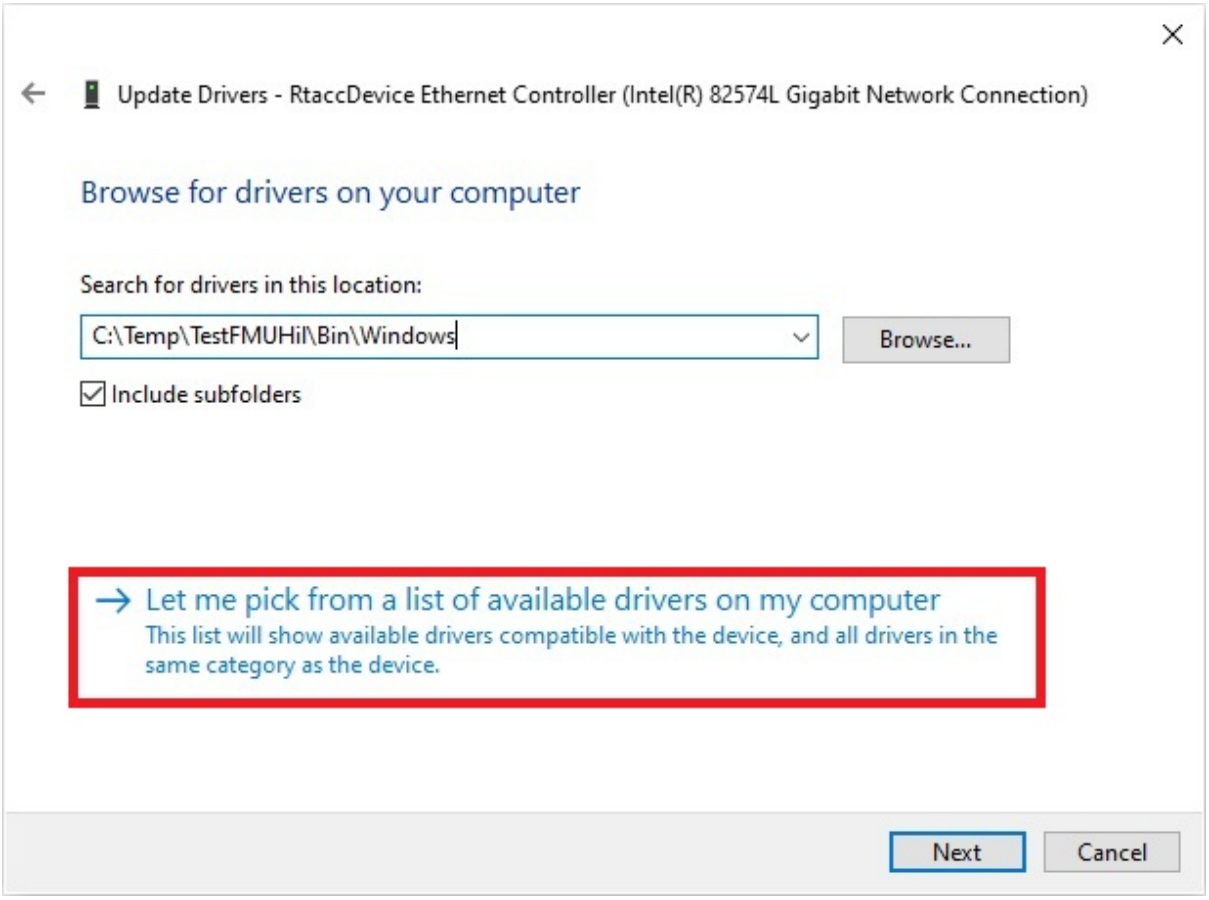
2. Right click on the RtaccDevice Ethernet Controller, then click on Properties:



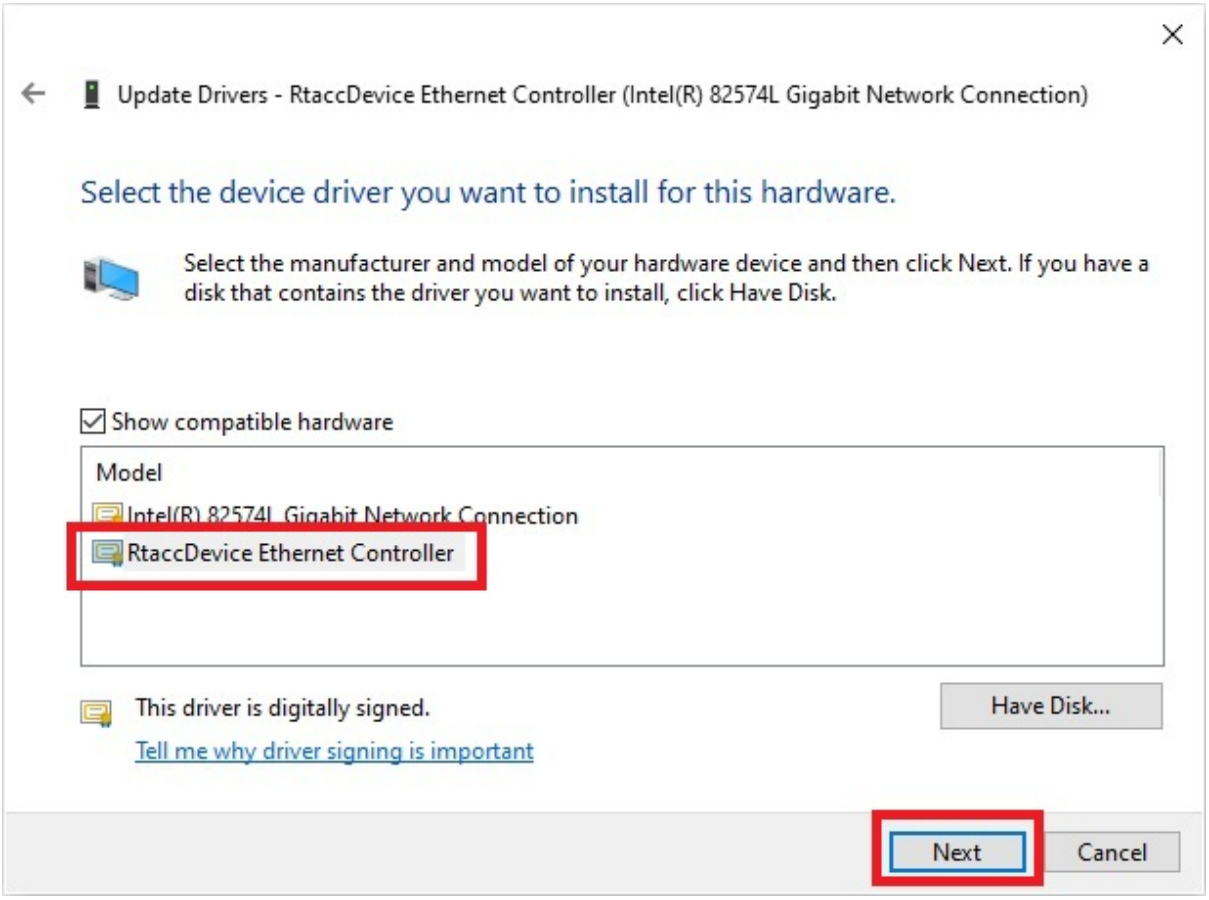
3. Switch to the tab “Driver”, then click the button “Update driver”:



4. Click on “Browse my computer for drivers”:



5. Click on “Let me pick from a list of available drivers on my computer”:



Select the network adapter and click “Next”.

The assignment is now reverted.

4.7 Xenomai

The system must be setup first the same way as for EC-Simulator for Linux, especially installation of the atemsys module and Real-time Ethernet Driver usage preparation.

See also:

Chapter *Linux*

The binaries are built using the following versions:

- **x64:**
  - Xenomai 3.0.2, tested on Linux Kernel 3.18.20 (Cobalt)
  - EVL r0.44 (Xenomai 4), tested on Linux Kernel 5.15.106
- **x86:**
  - Xenomai 2.6.2.1, tested on Linux Kernel 3.5.7
  - Xenomai 3.0.2, tested on Linux Kernel 3.18.20 (Cobalt) and 3.10.32-rt31 (Mercury)

4.7.1 OS compiler settings

Besides the general settings from *OS Compiler settings* the following settings are necessary to build the example application for Xenomai.

Extra include paths

```
<InstallPath>/SDK/INC/Xenomai
<InstallPath>/Examples/Common/Xenomai
```

Extra source paths

```
<InstallPath>/Examples/Common/Xenomai
<InstallPath>/Sources/OsLayer/Xenomai
```

Extra library paths to the main EtherCAT® components

- Xenomai 2 and 3:

```
<InstallPath>/SDK/LIB/Xenomai
```

- Xenomai 4:

```
<InstallPath>/SDK/LIB/Xenomai4
```

Extra libraries (in this order)

- Xenomai 2:

```
EcSimulatorRasServer EcSimulator pthread dl rt native xenomai
```

- Xenomai 3:

```
EcSimulatorRasServer EcSimulator pthread dl rt
```

`xeno-config --cflags` and `xeno-config --ldflags` of the Xenomai installation return the needed `CFLAGS` and `LDFLAGS`. If further information is needed, please refer to http://xenomai.org/ .

- Xenomai 4:

```
EcSimulatorRasServer EcSimulator pthread evl dl
```

5 Real-time Ethernet Driver

The EC-Simulator stack currently supports a variety of different Real-time Ethernet Driver modules, each contained in a single library file, which is loaded by the core library dynamically. Which library is actually loaded depends on the Link Layer parameters at runtime.

Real-time means operating directly on the register of the device set instead of using the operating system's native driver.

The principle of the Real-time Ethernet Driver selection is that the Ethernet Driver name (Ethernet Driver identification) is used to determine the location and name of a registration function called by EC-Simulator and registers function pointers that allow access to the Real-time Ethernet Driver functional entries.

The EtherCAT® Real-time Ethernet Driver will be initialized using a Real-time Ethernet Driver specific configuration parameter set. A pointer to this parameter set is part of EC-Simulator's initialization settings when calling the function *esInitSimulator()* .

EC-Simulator supports two Real-time Ethernet Driver operating modes:

- Interrupt mode all received Ethernet frames will be processed immediately in the context of the Real-time Ethernet Driver receiver task.
- Polling mode, the application must cyclically call *esExecJob()* with job *eUsrJob_ProcessAllRxFrames* in order to trigger EC-Simulator to call the Real-time Ethernet Driver receiver polling function and process received frames.

Important: In polling mode, the MainDevice cycle time must be at least two times higher than the simulator cycle time. E.g. if the simulator runs with 1 ms, the MainDevice cycle time must be at least 2 ms. If the Real-time Ethernet Driver is running in interrupt mode (non-standard), processing of received frames is done immediately after the frame is received.

Note: Real-time Ethernet Driver modules not listed here may be available if purchased additionally. Not all Real-time Ethernet Driver modules support interrupt mode.

5.1 Real-time Ethernet Driver initialization

The different Real-time Ethernet Driver modules are selected and parameterized by a Real-time Ethernet Driver specific structure.

The size of the Real-time Ethernet Driver parameter structure depends on the adapter type. All Real-time Ethernet Driver parameter structures start with *EC_T_LINK_PARMS* followed by adapter type specific parameters. The linkParms.dwSize must be set to sizeof(EC_T_LINK_PARMS_INTELGBE) or the used structure respectively. It may not be 0. The following example shows how to pass Real-time Ethernet Driver parameters to `emInitMaster()`:

```
/* master init parms */
EC_T_INIT_MASTER_PARMS oInitMasterParms;
OsMemset(&oInitMasterParms, 0, sizeof(EC_T_INIT_MASTER_PARMS));

/* simulator parms */
EC_T_LINK_PARMS_SIMULATOR oLinkParmsSimulator;
OsMemset(&oLinkParmsSimulator, 0, sizeof(EC_T_LINK_PARMS_SIMULATOR));

/* license Link Layer parms */
EC_T_LINK_PARMS_INTELGBE oLinkParmsIntelGbe;
```

(continues on next page)

(continued from previous page)

```
OsMemset(&oLinkParmsIntelGbe, 0, sizeof(EC_T_LINK_PARMS_INTELGBE));

/* identify simulator Link Layer in the common structure */
oLinkParmsSimulator.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_SIMULATOR;
oLinkParmsSimulator.linkParms.dwSize = sizeof(EC_T_LINK_PARMS_SIMULATOR);
OsSafeStrncpy(oLinkParmsSimulator.linkParms.szDriverIdent, EC_LINK_PARMS_IDENT_
↪SIMULATOR, EC_DRIVER_IDENT_NAME_SIZE);
oLinkParmsSimulator.linkParms.dwInstance = dwSimulatorId;
oLinkParmsSimulator.linkParms.eLinkMode = EcLinkMode_POLLING;

/* specific simulator parameters should be set here */
/* oLinkParmsSimulator.pbyCnfData = ENI/EXI; */

/* identify license Link Layer in the common structure */
oLinkParmsIntelGbe.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_INTELGBE;
oLinkParmsIntelGbe.linkParms.dwSize = sizeof(EC_T_LINK_PARMS_INTELGBE);
OsSafeStrncpy(oLinkParmsIntelGbe.linkParms.szDriverIdent, EC_LINK_PARMS_IDENT_
↪INTELGBE, EC_DRIVER_IDENT_NAME_SIZE);
oLinkParmsIntelGbe.linkParms.dwInstance = 1; /* instance ID of the adapter */
oLinkParmsIntelGbe.linkParms.eLinkMode = EcLinkMode_POLLING;

/* specific Link Layer parameters should be set here */
/* oLinkParmsIntelGbe.wRxBufferCnt = 128; */

/* pass license Link Layer parameters to simulator Link Layer */
oLinkParmsSimulator.apLinkParms[0] = (EC_T_LINK_PARMS*)&oLinkParmsIntelGbe;

/* more parameters should be set here */
oInitMasterParms.dwSignature = ATECAT_SIGNATURE;
oInitMasterParms.dwSize = sizeof(EC_T_INIT_MASTER_PARMS);

/* pass simulator Link Layer to master init parms */
oInitMasterParms.pLinkParms = (EC_T_LINK_PARMS*)&oLinkParmsSimulator;

/* initialize master */
dwRes = emInitMaster(dwInstanceId, &oInitMasterParms);
```

struct **EC_T_LINK_PARMS**

Public Members

EC_T_DWORD **dwSignature**
: [in] Signature of the adapter specific structure containing the *EC_T_LINK_PARMS* structure

EC_T_DWORD **dwSize**
: [in] Size of the adapter specific structure containing the *EC_T_LINK_PARMS* structure

EC_T_LOG_PARMS **LogParms**
: [in] Logging parameters

EC_T_CHAR **szDriverIdent**[EC_DRIVER_IDENT_NAME_SIZE]
: [in] Name of Link Layer module (driver identification) for Link Layer Selection

EC_T_DWORD **dwInstance**
: [in] Instance of the adapter. If EC_LINKUNIT_PCILOCATION is set: contains PCI address.

EC_T_LINKMODE **eLinkMode**
[in] Mode of operation

EC_T_CPUSET **cpuIstCpuAffinityMask**
[in] Interrupt service thread CPU affinity mask

EC_T_DWORD **dwIstPriority**
[in] Task priority of the interrupt service task (not used in polling mode)

EC_T_DWORD **dwIstStackSize**
[in] Task stack size

EC_T_DWORD **dwLinkSpeed**
[in] 10, 100, 1000 Mbit\s

EC_T_LINKLAYER_TIMINGTASK **oLinkLayerTimingTask**
[in] LinkLayer timimg task parameters

EC_T_CHAR **szLoadPath**[EC_DRIVER_PATH_SIZE]
[in] Path from which the libraries should be loaded

enum **EC_T_LINKMODE**
Values:

enumerator **EcLinkMode_UNDEFINED**
Link is in an undefined state, must be polling or interrupt

enumerator **EcLinkMode_INTERRUPT**
Link is in interrupt mode and is triggered by interrupts

enumerator **EcLinkMode_POLLING**
Link is in polling mode and is polled periodically

struct **EC_T_LINKLAYER_TIMINGTASK**

Public Members

EC_T_LINKLAYER_TIMING **eLinkLayerTiming**
[in] LinkLayer timing task mode

EC_T_DWORD **dwCycleTimeNsec**
[in] Cycle time between 2 pfnStartCycle calls in ns. Will be set by the master stack for the link layer.

EC_T_LINK_STARTCYCLE_CALLBACK **pfnStartCycle**
[in] Callback function called cyclically according to dwCycleTimeNsec

EC_T_VOID ***pvStartCycleContext**
[in] Context passed to each pfnStartCycle call

EC_T_DWORD **dwTtsSendOffsetUsec**
[in] Time between pfnStartCycle call and TTS frame transmission

EC_T_UINT64 **nSystemTime**
[in] System

enum **EC_T_LINKLAYER_TIMING**
Values:

enumerator **eLinkLayerTiming_Undefined**
Link Layer Timing is undefined must be TTS or TMR

enumerator **eLinkLayerTiming_TTS**
Real-time Ethernet Driver Time Triggered Send

enumerator **eLinkLayerTiming_TMR**
Real-time Ethernet Driver Timer

enum **EC_T_PHYINTERFACE**
Values:

enumerator **ePHY_UNDEFINED**
Undefined

enumerator **ePHY_FIXED_LINK**
No PHY access at all

enumerator **ePHY_MII**
MII 10 / 100 MBit

enumerator **ePHY_RMII**
Reduced MII, 10 / 100 MBit

enumerator **ePHY_GMII**
Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY_SGMII**
Serial (SERDES) Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY_RGMII**
Reduced Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY_OSDRIVER**
Get interface type from OS

enumerator **ePHY_RMII_50MHZ**
ePHY_RMII with 50 MHz clock mode

enumerator **ePhy_10GBASER**
10GBASE-R (64b/66b) serial interface, 10 GBit

enumerator **ePhy_25GBASER**
25GBASE-R (64b/66b) serial interface, 25 GBit

5.1.1 Real-time Ethernet Driver instance selection via PCI location

For some operating systems it is possible to address the Real-time Ethernet Driver instance using its PCI address as an alternative. To do this, EC_LINKUNIT_PCILOCATION (0x01000000) and the PCI location must be set as *EC_T_LINK_PARMS::dwInstance*.

On Linux the PCI address can be shown using e.g.:

```
$ lspci | grep Ethernet
```

```
$ 00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217-LM (rev 04)
$ 04:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
$ 05:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
```

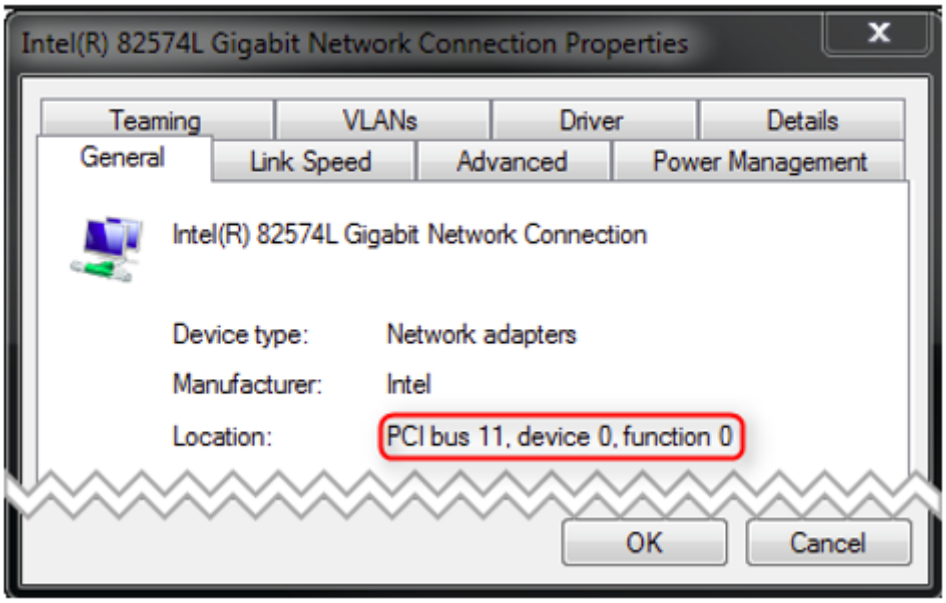
The format of *EC_T_LINK_PARMS::dwInstance* using PCI bus address is:

0x01bbddff

- *bb* Bus Number
- *dd* Device Number
- *ff* Function Number

```
EC_T_LINK_PARMS::dwInstance = 0x01001900; //"0000:00:19.0"
```

On Windows the integer value displayed in the properties dialog must be converted to HEX. E.g the number from the following dialog *(PCI bus 11, device 0, function 0)* corresponds to *0x010B0000* (bus *0x0B*).



5.2 Intel Pro/1000 - emllIntelGbe

The parameters to the Real-time Ethernet Driver Intel Pro/1000 are setup-specific. The function “CreateLinkParmsFromCmdLineIntelGbe” in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_INTELGBE**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_INTELGBE.

EC_T_WORD **wRxBufferCnt**
Receive buffer count, 0: default to 96

EC_T_WORD **wRxBufferSize**
Receive buffer size for a single Ethernet frame. 0: buffer optimized for standard Ethernet frame.

EC_T_WORD **wTxBufferCnt**
Transmit buffer count, 0: default to 96

EC_T_WORD **wTxBufferSize**
Transmit buffer size for a single Ethernet frame. 0: buffer optimized for standard Ethernet frame.

EC_T_BOOL **bDisableLocks**
Disable locks

EC_T_DWORD **dwAutoNegTimeout**
Timeout [ms] for auto negotiation

EC_T_BOOL **bNotUseDmaBuffers**
EC_FALSE: Use buffers from DMA for receive (default), EC_TRUE: use buffers from heap for receive. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_BOOL **bNoPhyCtrlOnConnect**
EC_TRUE: No PHY control (e.g. PHY reset, PHY PM settings, Gbits Ctrl) on link connection detected

NICs equipped with 82577, 82579 or 82567 may need HardCodedPhySettings. This must be set after *esInitSimulator()*, before using the NIC, e.g.:

```
{
    esIoCtl(dwSimulatorInstanceId, EC_IOCTL_LINKLAYER_MAIN | EC_LINKIOCTL_
↪FORCELINKMODE, (EC_T_BYTE*)EC_NULL + 0x20103, sizeof(EC_T_DWORD), EC_NULL, 0,␣
↪EC_NULL);
    OsSleep(1000);
}
```

5.2.1 TTS Feature

The IntelGbe Real-time Ethernet Driver can optionally use Time-Triggered Send (TTS) feature. Ethernet/EtherCAT® frames are sent and time stamped according to the NIC timer instead of the CPU timer. Which is usually more accurate.

See also:

EC_T_LINKLAYER_TIMINGTASK, *EC_T_LINKLAYER_TIMING*

5.2.2 Supported PCI devices

Intel PRO-1000 PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_I82540EM_DESKTOP** (0x8086, 0x100E)
- **PCI_DEVICE_I82545EM_COPPER** (0x8086, 0x100F)
- **PCI_DEVICE_I82546EB_COPPER_DUAL** (0x8086, 0x1010)
- **PCI_DEVICE_I82541EI_COPPER** (0x8086, 0x1013)
- **PCI_DEVICE_I82547GI_COPPER** (0x8086, 0x1019)
- **PCI_DEVICE_I82545GM_COPPER** (0x8086, 0x1026)
- **PCI_DEVICE_I82566MM** (0x8086, 0x1049)
- **PCI_DEVICE_I82566DM** (0x8086, 0x104A)
- **PCI_DEVICE_I82566MC** (0x8086, 0x104D)
- **PCI_DEVICE_N1E5132_SERVER** (0x8086, 0x105E)
- **PCI_DEVICE_I82547EI** (0x8086, 0x1075)
- **PCI_DEVICE_I82541GI_COPPER** (0x8086, 0x1076)
- **PCI_DEVICE_I82541GI_MOBILE** (0x8086, 0x1077)
- **PCI_DEVICE_I82541ER** (0x8086, 0x1078)
- **PCI_DEVICE_I82546GB_COPPER_DUAL** (0x8086, 0x1079)
- **PCI_DEVICE_I82541PI_DESKTOP** (0x8086, 0x107C)
- **PCI_DEVICE_I82572EI** (0x8086, 0x107D)
- **PCI_DEVICE_I82573E** (0x8086, 0x108B)
- **PCI_DEVICE_I82573** (0x8086, 0x108C)
- **PCI_DEVICE_I82573L** (0x8086, 0x109A)
- **PCI_DEVICE_I82571GB_QUAD** (0x8086, 0x10A4)
- **PCI_DEVICE_I82575_ZOAR** (0x8086, 0x10A7)
- **PCI_DEVICE_I82572GI** (0x8086, 0x10B9)
- **PCI_DEVICE_I82571GB_QUAD_2** (0x8086, 0x10BC)
- **PCI_DEVICE_I82566L** (0x8086, 0x10BD)
- **PCI_DEVICE_I82576** (0x8086, 0x10C9)
- **PCI_DEVICE_I82567V** (0x8086, 0x10CE)
- **PCI_DEVICE_I82574L** (0x8086, 0x10D3)
- **PCI_DEVICE_I82567LM3** (0x8086, 0x10DE)
- **PCI_DEVICE_I82577LM** (0x8086, 0x10EA)
- **PCI_DEVICE_I82577LC** (0x8086, 0x10EB)
- **PCI_DEVICE_I82578DM** (0x8086, 0x10EF)
- **PCI_DEVICE_I82578DC** (0x8086, 0x10F0)
- **PCI_DEVICE_I82567LM** (0x8086, 0x10F5)
- **PCI_DEVICE_I82567V3** (0x8086, 0x1501)
- **PCI_DEVICE_I82579LM** (0x8086, 0x1502)
- **PCI_DEVICE_I82579V** (0x8086, 0x1503)
- **PCI_DEVICE_I82576NS** (0x8086, 0x150A)
- **PCI_DEVICE_I82583V** (0x8086, 0x150C)
- **PCI_DEVICE_I82580_QUAD** (0x8086, 0x150E)
- **PCI_DEVICE_I350** (0x8086, 0x1521)
- **PCI_DEVICE_I82576_ET2** (0x8086, 0x1526)
- **PCI_DEVICE_I82580_QUAD_FIBRE** (0x8086, 0x1527)
- **PCI_DEVICE_I210AT** (0x8086, 0x1531)
- **PCI_DEVICE_I210AT_2** (0x8086, 0x1532)
- **PCI_DEVICE_I210_COPPER** (0x8086, 0x1533)
- **PCI_DEVICE_I210IT** (0x8086, 0x1535)
- **PCI_DEVICE_I210_SERDES** (0x8086, 0x1537)
- **PCI_DEVICE_I211AT** (0x8086, 0x1539)
- **PCI_DEVICE_I210_COPPER_FLASHLESS** (0x8086, 0x157B)
- **PCI_DEVICE_I210_BACKPLANE** (0x8086, 0x157C)
- **PCI_DEVICE_I217LM** (0x8086, 0x153A)
- **PCI_DEVICE_I217V** (0x8086, 0x153B)
- **PCI_DEVICE_I218LM** (0x8086, 0x155A)
- **PCI_DEVICE_I218V** (0x8086, 0x1559)
- **PCI_DEVICE_I218LM_2** (0x8086, 0x15A0)
- **PCI_DEVICE_I218V_2** (0x8086, 0x15A1)
- **PCI_DEVICE_I218LM_3** (0x8086, 0x15A2)
- **PCI_DEVICE_I218V_3** (0x8086, 0x15A3)
- **PCI_DEVICE_I219LM** (0x8086, 0x156F)
- **PCI_DEVICE_I219LM_2** (0x8086, 0x15B7)
- **PCI_DEVICE_I219LM_3** (0x8086, 0x15B9)
- **PCI_DEVICE_I219LM_4** (0x8086, 0x15D7)
- **PCI_DEVICE_I219LM_5** (0x8086, 0x15E3)
- **PCI_DEVICE_I219LM_6** (0x8086, 0x15BD)
- **PCI_DEVICE_I219LM_7** (0x8086, 0x15BB)
- **PCI_DEVICE_I219LM_8** (0x8086, 0x15DF)
- **PCI_DEVICE_I219LM_9** (0x8086, 0x15E1)
- **PCI_DEVICE_I219V** (0x8086, 0x1570)
- **PCI_DEVICE_I219V_2** (0x8086, 0x15B8)
- **PCI_DEVICE_I219V_4** (0x8086, 0x15D8)
- **PCI_DEVICE_I219V_5** (0x8086, 0x15D6)
- **PCI_DEVICE_I219V_6** (0x8086, 0x15BE)
- **PCI_DEVICE_I219V_7** (0x8086, 0x15BC)
- **PCI_DEVICE_I219V_8** (0x8086, 0x15E0)
- **PCI_DEVICE_I219V_9** (0x8086, 0x15E2)
- **PCI_DEVICE_I219LM_10** (0x8086, 0x0D4E)
- **PCI_DEVICE_I219V_10** (0x8086, 0x0D4F)
- **PCI_DEVICE_I219LM_11** (0x8086, 0x0D4C)
- **PCI_DEVICE_I219V_11** (0x8086, 0x0D4D)
- **PCI_DEVICE_I219LM_12** (0x8086, 0x0D53)
- **PCI_DEVICE_I219V_12** (0x8086, 0x0D55)
- **PCI_DEVICE_I219LM_18** (0x8086, 0x0DC5)
- **PCI_DEVICE_I219V_18** (0x8086, 0x0DC6)
- **PCI_DEVICE_I219LM_19** (0x8086, 0x0DC7)
- **PCI_DEVICE_I219V_19** (0x8086, 0x0DC8)
- **PCI_DEVICE_I219LM_13** (0x8086, 0x15FB)
- **PCI_DEVICE_I219V_13** (0x8086, 0x15FC)

- **PCI_DEVICE_I219LM_14** (0x8086, 0x15F9)
- **PCI_DEVICE_I219V_14** (0x8086, 0x15FA)
- **PCI_DEVICE_I219LM_15** (0x8086, 0x15F4)
- **PCI_DEVICE_I219V_15** (0x8086, 0x15F5)
- **PCI_DEVICE_I219LM_16** (0x8086, 0x1A1E)
- **PCI_DEVICE_I219V_16** (0x8086, 0x1A1F)
- **PCI_DEVICE_I219LM_17** (0x8086, 0x1A1C)
- **PCI_DEVICE_I219V_17** (0x8086, 0x1A1D)
- **PCI_DEVICE_I219LM_20** (0x8086, 0x550A)
- **PCI_DEVICE_I219V_20** (0x8086, 0x550B)
- **PCI_DEVICE_I219LM_21** (0x8086, 0x550C)
- **PCI_DEVICE_I219V_21** (0x8086, 0x550D)
- **PCI_DEVICE_I219LM_22** (0x8086, 0x550E)
- **PCI_DEVICE_I219V_22** (0x8086, 0x550F)
- **PCI_DEVICE_I219LM_23** (0x8086, 0x5510)
- **PCI_DEVICE_I219V_23** (0x8086, 0x5511)
- **PCI_DEVICE_I219LM_24** (0x8086, 0x57A0)
- **PCI_DEVICE_I219V_24** (0x8086, 0x57A1)
- **PCI_DEVICE_I225LM** (0x8086, 0x15F2)
- **PCI_DEVICE_I225V** (0x8086, 0x15F3)
- **PCI_DEVICE_I225I** (0x8086, 0x15F8)
- **PCI_DEVICE_I225K** (0x8086, 0x3100)
- **PCI_DEVICE_I225K_2** (0x8086, 0x3101)
- **PCI_DEVICE_I225LMVP** (0x8086, 0x5502)
- **PCI_DEVICE_I225IT** (0x8086, 0x0D9F)
- **PCI_DEVICE_I226LM** (0x8086, 0x125B)
- **PCI_DEVICE_I226V** (0x8086, 0x125C)
- **PCI_DEVICE_I226IT** (0x8086, 0x125D)

5.3 Intel Pro/100 - emIII8255x

The parameters to the Real-time Ethernet Driver Intel Pro/100 are setup-specific. The function `CreateLinkParmsFromCmdLinei8255x()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_I8255X**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_I8255X.

```
#include "EcLink.h"
EC_T_LINK_PARMS_I8255X oLinkParmsAdapter;

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_I8255X));
oLinkParmsAdapter.linkParms.dwSignature   = EC_LINK_PARMS_SIGNATURE_I8255X;
oLinkParmsAdapter.linkParms.dwSize        = sizeof(EC_T_LINK_PARMS_I8255X);
OsStrncpy(oLinkParmsAdapter.linkParms.szDriverIdent,
        EC_LINK_PARMS_IDENT_I8255X, MAX_DRIVER_IDENT_LEN - 1);
oLinkParmsAdapter.linkParms.dwInstance    = 1;
oLinkParmsAdapter.linkParms.eLinkMode     = EcLinkMode_POLLING;
oLinkParmsAdapter.linkParms.dwIstPriority = dwIstPriority;
```

5.3.1 Supported PCI devices

Intel PRO-100 PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_I82801DB** (0x8086, 0x103a)
- **PCI_DEVICE_I8255X** (0x8086, 0x1229)
- **PCI_DEVICE_I8255X_ER** (0x8086, 0x1209)
- **PCI_DEVICE_I8255X_VE** (0x8086, 0x1050)
- **PCI_DEVICE_I82562_VM** (0x8086, 0x1039)
- **PCI_DEVICE_I82559_ER** (0x8086, 0x2449)
- **PCI_DEVICE_I8255X_VE2** (0x8086, 0x27DC)
- **PCI_DEVICE_I82551_QM** (0x8086, 0x1059)
- **PCI_DEVICE_I8255X_VE3** (0x8086, 0x1092)

5.4 Broadcom Genet - emllBcmGenet

The parameters to the Real-time Ethernet Driver Broadcom® Genet are setup-specific. The function `CreateLinkParmsFromCmdLineBcmGenet()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_BCMGENET**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_BCMGENET.

EC_T_BCMGENET_TYPE **eSocType**
Broadcom processor type

EC_T_BOOL **bNotUseDmaBuffers**
EC_FALSE: Use buffers from DMA for receive (default), EC_TRUE: use buffers from heap for receive. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_PHYINTERFACE **ePhyInterface**
PHY interface type

enum **EC_T_BCMGENET_TYPE**
Values:

enumerator **eBCMGENET_BCM2711**
Broadcom BCM2711, Raspberry Pi 4

5.5 Broadcom NetXtreme - emllBcmNetXtreme

The parameters to the Real-time Ethernet Driver Broadcom® NetXtreme are setup-specific. The function `CreateLinkParmsFromCmdLineBcmNetXtreme()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Link Layer instance.

struct **EC_T_LINK_PARMS_BCMNETXTREME**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_BCMNETXTREME.

EC_T_DWORD **dwRxBuffers**
Receive buffer count. Must be a power of 2, maximum 1024.

EC_T_DWORD **dwTxBuffers**
Transmit buffer count. Must be a power of 2, maximum 1024.

5.5.1 Supported PCI devices

Broadcom 571x PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_BCM5717** (0x14E4, 0x1655)
- **PCI_DEVICE_BCM571X** (0x14E4, 0x1656)
- **PCI_DEVICE_BCM5719** (0x14E4, 0x1657)
- **PCI_DEVICE_BCM5720** (0x14E4, 0x1656)

5.6 Beckhoff CCAT - emllCCAT

The parameters to the Real-time Ethernet Driver CCAT are setup-specific. The function `CreateLinkParms-FromCmdLineCCAT()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance. Because the link status cannot be read quickly from a register of the adapter, it will not be automatically refreshed like by the other Real-time Ethernet Drivers.

struct **EC_T_LINK_PARMS_CCAT**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_CCAT.

EC_T_CCAT_TYPE **eCcatType**
CCAT connection type

EC_T_UINT64 **qwCcatBase**
Physical address of register block, only for eCCAT_EIM

EC_T_DWORD **dwCcatSize**
Size of register block, only for eCCAT_EIM

EC_T_DWORD **dwRxBufferCnt**
Receive buffer count, only for eCCAT_EIM

EC_T_DWORD **dwTxBufferCnt**
Transmit buffer count, only for eCCAT_EIM

enum **EC_T_CCAT_TYPE**
Values:

enumerator **eCCAT_PCI**
CCAT connected to PCI bus

enumerator **eCCAT_EIM**
CCAT connected via EIM. Used in ARM systems, no DMA

5.6.1 Supported PCI devices

Beckhoff CCAT PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_CCAT** (0x15EC, 0x2600)

5.7 Texas Instruments CPSW - emllCPSW

The parameters to the Real-time Ethernet Driver CPSW are setup-specific. The function `CreateLinkParmsFromCmdLineCPSW()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_CPSW**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_CPSW.

EC_T_CPSW_TYPE **eCpswType**
CPSW type

EC_T_DWORD **dwPhyAddr**
PHY address

EC_T_DWORD **dwPortPrio**
0 (lowest), 1 (highest)

EC_T_BOOL **bMaster**
EC_TRUE: Initialize MAC

EC_T_BOOL **bPhyRestartAutoNegotiation**
EC_TRUE: Restart auto negotiation on initialization

EC_T_PHYINTERFACE **ePhyInterface**
PHY connection type

EC_T_DWORD **dwRxInterrupt**
Receive interrupt number (IRQ)

EC_T_BOOL **bNotUseDmaBuffers**
Use buffers from DMA (EC_FALSE) or from heap for receive. AllocSend is not supported, when EC_TRUE. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_BOOL **bNoPhyAccess**
Don't use MDIO to set up the PHY

EC_T_BOOL **bAleBypass**
Enable bAleBypass, similar to promiscuous mode

EC_T_DWORD **dwExtendRxFrameLength**
Receive longer Rx Frames

enum **EC_T_CPSW_TYPE**
Values:

enumerator **eCPSW_AM33XX**
TI AM33xx (e.g. Beaglebone)

enumerator **eCPSW_AM387X**
TI DM814x/AM387x (e.g. Mistral/TI 814X/387X BASE EVM)

enumerator **eCPSW_AM437X**
TI AM437x

enumerator **eCPSW_AM57X**
TI AM57x

5.7.1 CPSW usage under Linux

Due to the lacking unbind-feature of the CPSW driver, the target's Kernel must not load the CPSW driver when starting. If the CPSW was built as a module, it can be renamed or removed to ensure it never gets loaded. If it was compiled into the Kernel, the Kernel needs to be recompiled without it.

It is possible to use one CPSW port for Linux kernel (TCP/IP) and another CPSW port for EC-Simulator. To do this, the CPSW kernel driver must be patched.

Currently following Linux versions are supported:

- linux-4.1.6 from TI Linux SDK 2.0
- linux-4.4.4-rt11 from Lenze
- linux-3.10.93-rt101 from Canon

Note: A patch for other Linux versions can also be created on request.

The patch needs:

- Linux kernel with enabled CPSW driver.
- Patch applied to Linux kernel.
- EC_ETHERNET_PORT defined according to target in cpsw.c and davinci_mdio.c files.
- Kernel must be rebuilt and installed

After that Linux will have only 1 Ethernet device, another can be used by EC-Simulator.

Note: EtherCAT® ports should be used as "slave" since "master" is the Linux driver.

5.8 Linux DPDK - emllDpdk

The Real-time Ethernet Driver emllDpdk is based on the Data Plane Development Kit (DPDK V23.11). See https://www.dpdk.org for more information. It does not need the atemsys driver and uses Ethernet adapters that are bound to the DPDK interface. The network interface must be bound according to the interface type PCI or SOC.

The parameters to emllDpdk are setup-specific. The function `CreateLinkParmsFromCmdLineDpdk()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize emllDpdk.

struct **EC_T_LINK_PARMS_DPDK**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_DPDK.

EC_T_DWORD **dwPortId**
DPDK port ID

EC_T_WORD **wRxBufferCnt**
Receive buffer count, 0: defaults to DPDK internal setting

EC_T_WORD **wTxBufferCnt**
Transmit buffer count, 0: default to DPDK internal setting

EC_T_BOOL **bDontCheckLinkStatus**
Don't check link status (forced)

EC_T_BOOL **bSetPromiscuousMode**
Change promiscuous mode setting

EC_T_BOOL **bEalInitDeinitByApp**
DPDK EAL layer is initialized and deinitialized by user App

Note:

- Root privileges are required.
- emllDpdk increases the stack size by DPDK_SEND_BURST_STACK_SIZE.

5.8.1 Linux System Requirements

- Kernel configuration

In the Fedora OS and other common distributions, such as Ubuntu, or Red Hat Enterprise Linux, the vendor supplied kernel configurations can be used to run most DPDK applications. For other kernel builds, options which should be enabled for DPDK include:

```
HUGETLBFS
PROC_PAGE_MONITOR support
```

- Required Tools, Libraries and further system requirements can be checked under https://doc.dpdk.org/guides/linux_gsg/sys_reqs.html

5.8.2 Huge pages setup

Create huge pages (not persistent)

```
$ mkdir -p /dev/hugepages
$ mountpoint -q /dev/hugepages || mount -t hugetlbfs nodev /dev/hugepages
$ echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

5.8.3 DPDK for PCI Network Adapter

First, the driver may need to be loaded, as it may be unloaded per default.

```
$ sudo modprobe uio_pci_generic
```

Next the network adapter needs to be bound with `dpdk-devbind.py`. Therefore, the PCI addresses have to be found out by calling

```
$ ./usertools/dpdk-devbind.py --status
```

The output will look like this (here the addresses of the PCI ports are 02:00.0 and 02:00.1, the card is an Intel X710, to be bound to the DPDK i40e driver):

```
Other Network adapters
======================
0000:02:00.0 'Ethernet Controller X710 for 10GBASE-T 15ff' unused=i40e,vfio-pci
0000:02:00.1 'Ethernet Controller X710 for 10GBASE-T 15ff' unused=i40e,vfio-pci
```

For each port to be bound to emllDpdk, call

```
$ ./usertools/dpdk-devbind.py --bind=uio_pci_generic <address>
```

Example:

```
$ ./usertools/dpdk-devbind.py --bind=uio_pci_generic 02:00.0
```

Check https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html for more information on the drivers.

5.8.4 DPDK for DPAA

Port specific numbering

The LS1046A Ethernet Port IDs within the Linux device tree (`dts`) may be non-linear, e.g. 1, 5, 6, 10, whereas the corresponding DPDK Port IDs are linear (0, 1, 2, …).

The following example demonstrates how the DPDK Port ID can be determined from the Ethernet Port ID within the Linux device tree:

DPDK	dts
0	1
1	5
2	6
3	10

Reassigning Ports to DPAA and Linux in DTB/DTS

The following changes make the LS1046A ports (i.e. ports 1, 5, 6, 10) available to the emllDpdk, the Linux boot file `/boot/fsl-ls1046a-frwy-sdk.dts/dtb`.

Note: ethernet@0 (Port 1) is known to be not assignable to emllDpdk for FRWY-LS1046A.

The section dpaa-extended-args must be extended:

```
chosen {
    stdout-path = "serial0:115200n8";
    name = "chosen";

    dpaa-extended-args {

        fman0-extd-args {
            cell-index = <0>;
            compatible = "fsl,fman-extended-args";
            dma-aid-mode = "port";

            fman0_rx0-extd-args {
                cell-index = <0>;
                compatible = "fsl,fman-port-1g-rx-extended-args";
                vsp-window = <8 0>;
            };

            fman0_tx0-extd-args {
                cell-index = <0>;
                compatible = "fsl,fman-port-1g-tx-extended-args";
            };

            fman0_rx1-extd-args {
                cell-index = <1>;
                compatible = "fsl,fman-port-1g-rx-extended-args";
                vsp-window = <8 0>;
            };

            fman0_tx1-extd-args {
                cell-index = <1>;
                compatible = "fsl,fman-port-1g-tx-extended-args";
            };
        };

        fman1-extd-args {
            cell-index = <1>;
            compatible = "fsl,fman-extended-args";
            dma-aid-mode = "port";
            fman1_rx0-extd-args {
                cell-index = <0>;
                compatible = "fsl,fman-port-1g-rx-extended-args";
                vsp-window = <8 0>;
            };

            fman1_tx0-extd-args {
                cell-index = <0>;
                compatible = "fsl,fman-port-1g-tx-extended-args";
            };

            fman1_rx1-extd-args {
                cell-index = <1>;
                compatible = "fsl,fman-port-1g-rx-extended-args";
```

(continues on next page)

(continued from previous page)

```
                vsp-window = <8 0>;
            };

            fman1_tx1-extd-args {
                cell-index = <1>;
                compatible = "fsl,fman-port-1g-tx-extended-args";
            };
        };
    };
};
```

Furthermore some or all ports must be assigned to DPDK instead of the native Linux driver in the section `fsl,dpaa` as follows:

```
fsl,dpaa {
    compatible = "fsl,ls1046a\\0fsl,dpaa\\0simple-bus";
    dma-coherent;
    phandle = <0x85>;

    /* assign Port 1 (ethernet@0) to emllDpdk (all ports assigned to emllDpdk must
↪be contiguous at the start) */

    ethernet@0 {
        compatible = "fsl,dpa-ethernet-init";
        fsl,fman-mac = <0x34>;
        dma-coherent;

        fsl,bman-buffer-pools = <0x35 0x36 0x37>;
        fsl,qman-frame-queues-rx = <0x50 0x01 0x51 0x01>;
        fsl,qman-frame-queues-tx = <0x70 0x01 0x71 0x01>;
    };

    /* port listed in BSP, but not implemented */

    ethernet@1 {
        compatible = "fsl,dpa-ethernet";
        fsl,fman-mac = <0x38>;
        dma-coherent;
        status = "disabled";
    };

    ...

    /* assign Port 10 (ethernet@9) to the native Linux driver */

    ethernet@9 {
        compatible = "fsl,dpa-ethernet";
        fsl,fman-mac = <0x3e>;
        dma-coherent;
    };
    ...
};
```

5.8.5 DPDK for ENETC4

DPDK for ENETC4 on the i.MX95 SoC currently requires a specifically customized DPDK version from NXP. Therefore, a specially created version `libemllDpdkEnetc4.so` exists which is linked against NXP DPDK 22.11.

To use this version of emllDpdk, the *EC_T_LINK_PARMS::szDriverIdent* of *EC_T_LINK_PARMS_DPDK::linkParms* must be set to `EC_LINK_PARMS_IDENT_DPDK_ENETC4`:

```
EC_T_LINK_PARMS_DPDK* pLinkParmsAdapter = EC_NULL;
/* ... */
OsStrcpy(pLinkParmsAdapter->linkparms.szDriverIdent, EC_LINK_PARMS_IDENT_DPDK_
↪ENETC4);
```

See also:

`CreateLinkParmsFromCmdLineDpdk()` in `EcSelectLinkLayer.cpp`

Additional requirements

- Kernel Module `kpage_ncache.ko` https://github.com/nxp-qoriq/dpdk-extras
- *devlink* tool from iproute2 https://github.com/mikedanese/iproute2/tree/master/devlink
- NXP DPDK 22.11 https://github.com/NXP/dpdk

Setting up the ENETC4 Ethernet interfaces

The following steps describe the creation of VF (Virtual Function) and the binding of DPDK to it.

1. Load Kernel modules

   ```
   $ modprobe uio_pci_generic
   $ modprobe kpage_ncache
   ```

2. Setup hugepages

   ```
   $ echo 448 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

3. Identify the PF(Physical Function) PCI address

   ```
   $ dpdk-devbind.py --status
   ```

   The devices are displayed in the output under *Network devices using kernel driver*; in this case, *0001:00:00.0* is used

   ```
   Network devices using kernel driver
   ===================================
   0001:00:00.0 'Device e101' if=end0 drv=fsl_enetc4 unused=
   0001:00:08.0 'Device e101' if=end1 drv=fsl_enetc4 unused= *Active*
   ```

4. Configure queues. Set the ring count on the PF to 1 for VF0 and VF1 and reload to apply the new configuration

   ```
   devlink dev param set pci/0001:00:00.0 name si_num_rings cmode driverinit
   ↪  value 0101
   devlink dev reload pci/0001:00:00.0
   ```

5. Create Virtual Function VF0

   ```
   echo 1 > /sys/bus/pci/devices/0001 00 00.0/sriov_numvfs
   ```

6. Identify the VF0 PCI address

   ```
   $ dpdk-devbind.py --status
   ```

   The newly created devices are displayed in the output under *Other Network devices*; in this case, *0001:00:02.0*

```
Other Network devices
=====================
0001:00:02.0 'Device ef00' unused=uio_pci_generic
0001:00:10.0 'Device e101' unused=fsl_enetc4,uio_pci_generic
```

7. Bind VF0 to DPDK

   ```
   dpdk-devbind.py -b uio_pci_generic 0001:00:02.0
   ```

8. Enable trust for VF0

   ```
   ip link set end0 vf 0 trust on
   ```

See also:

NXP RM00293

5.8.6 Limitations

If the adapter needs to be used multiple times, for cable redundancy or calling `emInitMaster()` multiple times, the application must set *`EC_T_LINK_PARMS_DPDK::bEalInitDeinitByApp`* = EC_TRUE and call *rte_eal_init()* and *rte_eal_cleanup()* itself.

To initialize DPDK following call is needed:

```
rte_eal_init();
```

To deinitialize DPDK following calls are needed:

```
rte_eth_dev_close(dwPortId);
rte_eal_cleanup();
```

5.9 DW3504 - emllDW3504

The parameters to the Real-time Ethernet Driver Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC (DW3504) are setup-specific. The function `CreateLinkParmsFromCmdLineDW3504()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **`EC_T_LINK_PARMS_DW3504`**

Public Members

EC_T_LINK_PARMS **`linkParms`**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_DW3504.

EC_T_DWORD **`dwPhyAddr`**
PHY address

EC_T_DWORD **`dwRegisterBasePhys`**
Physical base address of register block (8k)

EC_T_DW3504_TYPE **`eDW3504Type`**
System on Chip type

EC_T_PHYINTERFACE **ePhyInterface**
PHY connection type

EC_T_BOOL **bNotUseDmaBuffers**
EC_FALSE: Use buffers from DMA for receive (default), EC_TRUE: use buffers from heap for receive. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_DWORD **dwTxDmaDesCnt**
Transmit DMA descriptor buffer count. Must be a power of 2, maximum 256.

EC_T_DWORD **dwRxDmaDesCnt**
Receive DMA descriptor buffer count. Must be a power of 2, maximum 256.

EC_T_BOOL **bNotUseCacheSync**
Default use of CacheSync EC_FALSE, don't call CacheSync on older systems EC_TRUE

EC_T_BOOL **bUsePhyLib**
Use PhyLib instead of Legacy PHY handling for eDW3504_CycloneV, eDW3504_LCES1 or eDW3504_RZN1, for all others the PhyLib is mandatory

EC_T_BOOL **bNoPhyAccess**
Don't use MDIO to set up the PHY

EC_T_DWORD **dwRxInterrupt**
Receive interrupt number (IRQ)

enum **EC_T_DW3504_TYPE**
Values:

enumerator **eDW3504_CycloneV**
MAC on Cyclone V SoC

enumerator **eDW3504_LCES1**
MAC on LCES1 SoC

enumerator **eDW3504_RZN1**
MAC on Renesas RZN1

enumerator **eDW3504_STM32MP15x**
MAC on STM32MP15x

enumerator **eDW3504_ATOM**
MAC on Atom 6000

enumerator **eDW3504_STM32MP13x**
MAC on STM32MP13x

enumerator **eDW3504_RK3328**
MAC on Rockchip 3328 - Rock64

enumerator **eDW3504_RK3399**
MAC on Rockchip 3399 - Orange Pi 4

enumerator **eDW3504_RK3588S**
MAC on Rockchip 3588s - Orange Pi 5

enumerator **eDW3504_RK3568**
MAC on Rockchip 3568 - Radxa Rock3 a

enumerator **eDW3504_SemidriveD9**
MAC on Semidrive D9

enumerator **eDW3504_RK3576**
MAC on Rockchip 3576 - Tronlong TL-3576-EVM

enumerator **eDW3504_RK3562**
MAC on Rockchip 3562 - Embedfire LubanCat3

enumerator **eDW3504_AllWinnerT536**
MAC on AllWinner T536

enumerator **eDW3504_RZN2**
MAC on Renesas RZN2

enumerator **eDW3504_RK3506**
MAC on Rockchip 3506

enumerator **eDW3504_IQ9075**
MAC on Qualcomm Dragonwing IQ-9075 EVK

enumerator **eDW3504_NvidiaJetsonAgxOrin**
MAC on NVIDIA Jetson AGX Orin Dev Kit

5.9.1 Supported PCI devices

Synopsis DW3504 PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_INTEL_EHL_DWMAC_RGMII_1** (0x8086, 0x4BB0)
- **PCI_DEVICE_INTEL_EHL_DWMAC_RGMII_2** (0x8086, 0x4BA0)
- **PCI_DEVICE_INTEL_EHL_DWMAC_RGMII_3** (0x8086, 0x4B30)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_1** (0x8086, 0x4B32)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_2** (0x8086, 0x4BB1)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_3** (0x8086, 0x4BA1)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_4** (0x8086, 0x4B31)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_5** (0x8086, 0x4BA2)
- **PCI_DEVICE_INTEL_EHL_DWMAC_SGMII_6** (0x8086, 0x4BB2)
- **PCI_DEVICE_INTEL_TGL_DWMAC_SGMII_1** (0x8086, 0x43A2)
- **PCI_DEVICE_INTEL_TGL_DWMAC_SGMII_2** (0x8086, 0x43AC)
- **PCI_DEVICE_INTEL_TGL_DWMAC_SGMII_3** (0x8086, 0xA0AC)
- **PCI_DEVICE_INTEL_ADLS_DWMAC_SGMII_1** (0x8086, 0x7AAC)
- **PCI_DEVICE_INTEL_ADLS_DWMAC_SGMII_2** (0x8086, 0x7AAD)

5.10 Freescale TSEC / eTSEC - emllETSEC

The parameters to the Real-time Ethernet Driver ETSEC are setup-specific. The function `CreateLinkParmsFromCmdLineETSEC()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_ETSEC**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_ETSEC.

EC_T_DWORD **dwRegisterBase**
Physical base address of register block (4k)

EC_T_DWORD **dwLocalMdioBase**
Physical base address of local MDIO register block (4k). For the eTSEC V1 or TSEC this is the same as dwRegisterBase, for the eTSEC V2 it's not.

EC_T_DWORD **dwPhyMdioBase**
Physical base address of MDIO register block (4k). This is the MDIO base of the (e)TSEC where the PHY (MII bus) is physically connected to (MII interface is shared by (e)TSEC's).

EC_T_DWORD **dwPhyAddr**
PHY address on MII bus. ETSEC_FIXED_LINK if fixed link configuration.

EC_T_DWORD **dwTbiPhyAddr**
Address of internal TBI PHY. Any address from [0..31] can be used here, but the address shouldn't collide with any external PHY connected to the external MII bus.

EC_T_DWORD **dwFixedLinkVal**
Only evaluated if dwPhyAddr == FIXED_LINK. Set to one of the ETSEC_LINKFLAG_* macros. I.e. ETSEC_LINKFLAG_1000baseT_Full.

EC_T_BYTE **abyStationAddress**[6]
MAC address

EC_T_VOID ***oMiiBusMtx**
This mutex protects the access to the (shared) MII bus. Set to 0 if mutex shouldn't be used. The MII bus is shared between eTSEC instances. So this mutex should be created once and assigned here for all Linklayer instances.

EC_T_DWORD **dwRxInterrupt**
Receive interrupt number (IRQ)

EC_T_BOOL **bNotUseDmaBuffers**
EC_FALSE: Use buffers from DMA for receive (default), EC_TRUE: use buffers from heap for receive. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_ETSEC_TYPE **eETSECType**
System on Chip type

EC_T_BOOL **bMaster**
Full control over the MAC and need to initialize MAC and the connections to the PHYs

enum **EC_T_ETSEC_TYPE**
Values:

enumerator **eETSEC_P2020RDB**
MAC on Freescale P2020

enumerator **eETSEC_TWRP1025**
MAC on Freescale TWRP1025

enumerator **eETSEC_ISTMPC8548**
MAC on Freescale ISTMPC8548

enumerator **eETSEC_XJ_EPU20C**
MAC on Freescale XJ EPU20C

enumerator **eETSEC_TWRLS1021A**
MAC on Freescale TWRLS1021A

enumerator **eETSEC_TQMLS_LS102XA**
MAC on Freescale TQMLS LS102XA

5.10.1 ETSEC supported MAC's

- TSEC (not tested): Legacy hardware. Should be supported, because eTSEC is compatible to TSEC if the enhanced functionality is not used.
- eTSEC v1 (tested): This chip is used for QorIQ (i.e. P2020E) and PowerQUICC devices (i.e. MPC8548). It has 4k of IO memory.
- eETSEC v2, also called vETSEC, v read as "virtualization" (tested): This chip is used for newer QorIQ devices (i.e. P1020). It has 12k of IO memory (4k MDIO, 4k Register group0, 4k Register group1)

5.10.2 Shared MII bus

The driver will access the Ethernet PHY for the following reasons:

- Check for link (or timeout), if the driver instance is opened.
- Configure MAC according to the auto-negotiated PHY speed (mandatory).
- Check link (and reconfigure MAC) during cyclic run. Therefore `EC_LINKIOCTL_UPDATE_LINKSTATUS` should not be called explicitly in parallel!

Note: The external PHYs are connected physically to the MII bus of the first eTSEC (and/or eTSEC3, depending on SoC type). From SoC reference manuals: "14.5.3.6.6 MII Management Configuration Register (MIIMCFG) … Note that MII management hardware is shared by all eTSECs. Thus, only through the MIIM registers of eTSEC1 can external PHYs be accessed and configured."

That means that the acontis TSEC / eTSEC driver will also mmap the register set of the corresponding eTSEC. The following initialization parameters are used to specify the MII settings:

1. Memory map of eTSEC which will manage the MII bus (connection of external PHY's):

```
poDrvSpecificParam->dwPhyMdioBase = dwCcsrbar + 0x24000;
```

1. Dummy address assigned to internal TBI PHY. Use any address (from 0 .. 31) which will not collide with any of the physical PHY's addresses:

```
poDrvSpecificParam->dwTbiPhyAddr = 16;
```

5.10.3 Locking

The optional lock is acquired each time the MDIO register (specified by poDrvSpecificParam->dwPhyMdioBase) are accessed:

```
poDrvSpecificParam->oMiiBusMtx = EC_NULL;

/* implement locking by using return value of LinkOsCreateLock(eLockType_DEFAULT);␣
↪*/
```

5.10.4 Link check

The driver's API function EcLinkGetStatus (pfEcLinkGetStatus) is called by the EC-Simulator stack. On eTSEC the link status can't be obtained directly by reading eTSEC registers without access to the MII bus (Use mutex, poll for completion). Accessing the bus would violate timing constraints and is therefore not possible.

The following IOCTL updates the link status and accesses the PHY. The IOCTL is blocking and may therefore not be called from the JobTask's context. I.e. use:

```
dwRes = esIoCtl(dwSimulatorInstanceId, EC_IOCTL_LINKLAYER | EC_LINKIOCTL_UPDATE_
↪LINKSTATUS, EC_NULL);
```

EcLinkGetStatus always returns the last known link status.

5.10.5 Fixed Link

PHY access can be effectively disabled entirely to avoid concurrent access if link speed and mode are defined to be fixed. This functionality is mainly provided for L2-Switch-IC's like Vertesse VSC7385 which haven't any PHY and are attached to the eTSEC MAC with fixed speed and mode.

The driver's open function will not wait until the link is up on EC-Simulator start up. Auto-negotiation of the following PHYs is not affected by this parameter and still active. There is no forced link and no PHY access at all.

Parameters for fixed link:

```
pETSECParam->dwPhyAddr      = ETSEC_FIXED_LINK;
pETSECParam->dwFixedLinkVal = ETSEC_LINKFLAG_1000baseT_Full | ETSEC_LINKFLAG_
↪LINKOK;
```

5.11 Freescale FslFec - emllFslFec

The parameters to the Real-time Ethernet Driver FslFec are setup-specific. The function `CreateLinkParmsFromCmdLineFslFec()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_FSLFEC**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_FSLFEC

EC_T_DWORD **dwRxBuffers**
Receive buffer count

EC_T_DWORD **dwTxBuffers**
Transmit buffer count

EC_T_FEC_TYPE **eFecType**
System on Chip type

EC_T_PHYINTERFACE **ePhyInterface**
PHY interface type

EC_T_BOOL **bUseDmaBuffers**
Use buffers from DMA (EC_TRUE) or from heap for receive and AllocSend not supported (EC_FALSE)

EC_T_DWORD **dwPhyAddr**
PHY Address

EC_T_BOOL **bNoPinMuxing**
No clock configuration and pin muxing

EC_T_BOOL **bDontReadMacAddr**
Read of MAC address disabled

EC_T_DWORD **dwRxInterrupt**
Receive interrupt number (IRQ)

EC_T_BOOL **bNoPhyAccess**
EC_FALSE: Link layer should initialize PHY and read link status (connected/disconnected). EC_TRUE: Client is responsible of PHY initialization and clock initialization

EC_T_BOOL **bFlushFramesRequired**
EC_FALSE: flush cyclic / acyclic frames queued at link layer

enum **EC_T_FEC_TYPE**
Values:

enumerator **eFEC_IMX25**
MAC on Freescale i.MX25 (ARM9; ARMv5)

enumerator **eFEC_IMX28**
MAC on Freescale i.MX28 (ARM9; ARMv5)

enumerator **eFEC_IMX53**
MAC on Freescale i.MX53 (ARM Cortex-A8; ARMv7-a)

enumerator **eFEC_IMX6**
MAC on Freescale i.MX6 (ARM Cortex-A9 Single/Dual/Quad; ARMv7-a)

enumerator **eFEC_VF6**
MAC on Freescale VYBRID VF6xx (ARM Cortex-A5 + Cortex-M4)

enumerator **eFEC_IMX7**
MAC on Freescale i.MX7 (ARM Cortex-A9 Single/Dual/Quad; ARMv7-a)

enumerator **eFEC_IMX8**
MAC on Freescale i.MX8 (ARM Cortex-A72/A53 Single/Dual/Quad; ARMv8-a)

enumerator **eFEC_IMX8M**
MAC on Freescale i.MX8M Mini/Nano/Plus (ARM Cortex-A53 Single/Dual/Quad; ARMv8-a)

enumerator **eFEC_IMXRT1064**
MAC on NXP i.MX RT 1064 (ARM Cortex-M7)

enumerator **eFEC_IMX9**
MAC on NXP i.MX93 (ARM Cortex-A55/M33 Single/Dual)

enumerator **eFEC_IMX8MP**
MAC on NXP i.MX8MPLUS

5.12 Cadence GEM/MACB - emllGEM

The parameters to the Real-time Ethernet Driver GEM are setup-specific. The function `CreateLinkParmsFromCmdLineGEM()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_GEM**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_GEM.

EC_T_GEM_RXSOURCE **eRxSource**
Source of RX clock, control and data signals

EC_T_DWORD **dwPhyAddr**
PHY address

EC_T_DWORD **dwRxInterrupt**
Receive interrupt number (IRQ)

EC_T_BOOL **bUseDmaBuffers**
Use buffers from DMA (EC_TRUE) or from heap for receive. AllocSend is not supported when EC_FALSE.

EC_T_BOOL **bNoPhyAccess**
EC_FALSE: Link layer should initialize PHY and read link status (connected/disconnected). EC_TRUE: Client is responsible of PHY initialization and clock initialization.

EC_T_BOOL **bUseGmiiToRgmiiConv**
Use XILINX GMIITORGMII Converter (EC_TRUE)

EC_T_DWORD **dwConvPhyAddr**
PHY address used to communicate with converter. In Linux doc it is named “reg”.

EC_T_DWORD **dwTxDmaDesCnt**
Transmit DMA descriptor buffer count. Must be a power of 2, maximum 256.

EC_T_DWORD **dwRxDmaDesCnt**
Receive DMA descriptor buffer count. Must be a power of 2, maximum 256.

EC_T_GEM_TYPE **eGemType**
System on Chip type

EC_T_PHYINTERFACE **ePhyInterface**
PHY connection type

EC_T_BOOL **bNoPinMuxing**
No clock configuration and pin muxing

EC_T_GEM_CLK_DIV_TYPE **eClkDivType**
Change Ref Clock settings

EC_T_BOOL **bNotUseCacheSync**
Default use of CacheSync EC_FALSE, don’t call CacheSync on older systems EC_TRUE

enum **EC_T_GEM_RXSOURCE**
Values:

enumerator **eGemRxSource_MIO**
MIO as source for RX clock, control and data signals

enumerator **eGemRxSource_EMIO**
EMIO as source for RX clock, control and data signals

enum **EC_T_GEM_TYPE**
Values:

enumerator **eGemType_Zynq7000**
Xilinx Zynq 7000

enumerator **eGemType_ZynqUltrascale**
Xilinx Zynq Ultrascale

enumerator **eGemType_BCM2712**
Broadcom BCM2712, Raspberry Pi 5

enumerator **eGemType_PolarFire**
Microchip PolarFire SoC

5.13 INtime HPE - emllHPE

emllHPE is part of EC-Simulator for INtime. emllHPE uses the INtime High-Performance Ethernet (HPE) interface. The parameters to emllHPE are setup-specific. The function `CreateLinkParmsFromCmdLineHPE()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_HPE**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_HPE.

EC_T_CHAR **szAdapterName**[EC_HPE_ADAPTER_NAME_SIZE]
Name of Ethernet Interface, e.g. “ie1g0”

EC_T_DWORD **dwRxBufferCnt**
Receive buffer count

EC_T_DWORD **dwTxBufferCnt**
Transmit buffer count

EC_T_BOOL **bDisableForceBroadcast**
Don’t change target MAC address to FF:FF:FF:FF:FF:FF

5.14 Texas Instruments ICSS - emllICSS

The parameters to the Real-time Ethernet Driver ICSS are setup-specific. The function `CreateLinkParmsFromCmdLineICSS()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_ICSS**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_ICSS.

EC_T_LINK_ICSS_BOARD **eBoardType**
TI System on Chip board type

EC_T_BOOL **bMaster**
Initialize whole PRUSS subsystem, not only port. This flag is always required when the link layer is used on a single ICSS port. This flag is also required, when the link layer is used in “Redundancy mode” and two ICSS ports are used. In this case, first port should be master, and second port should be slave.

EC_T_BOOL **bNoMacAddr**
EC_TRUE: No MAC address registers access

EC_T_DWORD **dwPhyAddr**
PHY address

EC_T_PHYINTERFACE **ePhyInterface**
PHY connection type

EC_T_BOOL **bNoPhyReset**
No hardware reset of the PHY

EC_T_BOOL **bUseAllSendQueues**
Use the additional 3 queues with lower priority to send more frames per cycle

EC_T_BOOL **bLegacyFirmware**
For am57xx use legacy ICSS firmware from pdk_am57xx_1_0_6, instead of pdk_am57xx_1_0_17 with patch for Rx error issue, see https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1022410/am5746-rx_error_offset-conditions/3788558#3788558

enum **EC_T_LINK_ICSS_BOARD**
Values:

enumerator **EcLinkIcssBoard_Unsupported**

enumerator **EcLinkIcssBoard_am572x**
TI AM572x

enumerator **EcLinkIcssBoard_am571x**
TI AM571x

enumerator **EcLinkIcssBoard_am3359**
TI AM3359

enumerator **EcLinkIcssBoard_am572x_emerson**
TI AM572x on Emerson board

enumerator **EcLinkIcssBoard_am574x**
TI AM574x

5.14.1 TTS Feature

Real-time Ethernet Driver PRU ICSS can optionally use Time-Triggered Send feature https://www.ti.com/lit/pdf/tidubz1

To test it, you need to build a demo application with INCLUDE_TTS macro. Additionally, you need to set the bTts flag and configure other tts parameters in *EC_T_LINK_PARMS_ICSS* structure. Please note, we have already TTS Demo applications for some of the operating systems (for ex. Linux and TI RTOS).

dwTtsSendTimeUsec time is determined experimentally. It depends on how long your own real project prepares cyclic and acyclic frames to be sent in the current cycle.

The main purpose of the TTS feature is to reduce jitter to 40 ns (nanoseconds). To measure jitter accurately you need to have special software and hardware. For example:

- Old version of Wire Shark, ex. 1.8.4
- Dissect plugin for Wire Shark (this plugin is available only for this version of WireShark)
- ET2000 device to insert accurate timestamps with nanoseconds resolution.

Details can be found here: https://infosys.beckhoff.com/index.php?content=../content/1031/et2000/1309654283.html&id=

5.14.2 TI AM335x ICEV2

After the two 100 MBit ports have been deactivated, there are no longer any Ethernet ports that can be used for TCP/IP. The board cannot work in mixed mode, i.e. there is no CPSW+ICSS support. It is also necessary to configure the board to start in ICSS rather than CPSW mode. Set both jumpers on the board to ICSS mode.

See also:

http://processors.wiki.ti.com/index.php/AM335x_Industrial_Communication_Engine_EVM_Rev2_1_HW_User_Guide

5.14.3 TI AM57xx IDK

After the four 100 MBit ports of the ICSS have been deactivated, the other two 1 GBit ports (CPSW) remain active and can be used for other purposes (e.g. TCP / IP).

5.14.4 AM5728 IDK and AM5718 IDK boards and Technical Limitations

The main difference between these two boards is number of available ICSS ports. AM5728 IDK supports only two 100 Mbit ports: port 3 and 4. It is a technical limitation of this board. On AM5718 IDK all four 100 Mbit ports are available for EtherCAT® purposes.

Note: Real-time Ethernet Driver PRUICSS can use at most 2 ports together (in redundancy mode) and these two ports should correspond to the same PRUSS. I.e. Port 3 and 4 OR Port 1 and 2, but not Port 1 and 4, Port 1 and 3 and etc. This technical limitation exists, because PRU firmware for PRU0 and PRU1 uses the same memory areas of OCMC Memory. In the future, this limitation can be removed.

5.15 Windows NDIS - emllNdis

As default EC-Simulator for Windows contains `emllNdis.dll` to use a native Windows driver for EtherCAT®.

The acontis ECAT Protocol Driver is needed to use the Ethernet Driver NDIS and can be installed from

- `Bin/Windows/x64/EcatNdisSetup-x86_64Bit.msi` or
- `Bin/Windows/x86/EcatNdisSetup-x86_32Bit.msi`

respectively depend on the Windows Operating System Type of 64 Bit or 32 Bit.

IPv4 must be installed for the network adapter as the Ethernet Driver NDIS uses the IP address to identify the network adapter.

The parameters to the Ethernet Driver NDIS are setup-specific. The function `CreateLinkParmsFromCmdLineNDIS()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Ethernet Driver instance.

struct **EC_T_LINK_PARMS_NDIS**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_NDIS.

EC_T_CHAR **szAdapterName**[EC_NDIS_ADAPTER_NAME_SIZE]
ServiceName of network adapter, see HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards in registry (zero terminated)

EC_T_BYTE **abyIpAddress**[4]
IP address of network adapter

EC_T_BOOL **bDisablePromiscuousMode**
Disable adapter promiscuous mode

EC_T_BOOL **bDisableForceBroadcast**
Don't change target MAC address to FF:FF:FF:FF:FF:FF

In case of problems while using the Ethernet Driver, it is advised to set the windows registry entry DontSetPromiscuousMode of the ECAT NDIS Protocol driver. This option is available since V3.1.3.02 of the driver. This can be done through the following steps:

- Install ECAT NDIS Protocol driver (V3.1.3.02 or newer version)
- Open the registry editor
- Switch to Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ecatndis, or just look for Ecatndis in the editor
- Create a new DWORD entry named DontSetPromiscuousMode
- Set the value of DontSetPromiscuousMode to 1
- Close the registry editor and restart your computer

5.16 Windows WinPcap - emllPcap

An Ethernet Driver based on the WinPcap library is shipped with the EC-Simulator. This Ethernet Driver is implemented using a network filter driver that enables the software to send and receive raw Ethernet frames. Using this Ethernet Driver any Windows standard network drivers can be used. The Windows network adapter card has to be assigned a unique IP address (private IP address range). This IP address is used by the EtherCAT® WinPcap Ethernet Driver driver to select the appropriate adapter.

It is recommended to use a separate network adapter to connect EtherCAT® devices. If the network adapter is used for both EtherCAT® and the local area network, it may significantly impact local area network operation. A static private IP address (e.g., 192.168.x.y) must be assigned to the EtherCAT® network adapter.

The parameters to the Ethernet Driver WinPcap are setup-specific. The function `CreateLinkParmsFromCmdLineWinPcap()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Ethernet Driver instance.

struct **EC_T_LINK_PARMS_WINPCAP**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_WINPCAP.

EC_T_BYTE **abyIpAddress**[4]
IP address

EC_T_CHAR **szAdapterId**[MAX_LEN_WINPCAP_ADAPTER_ID]
Adapter ID, format: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}

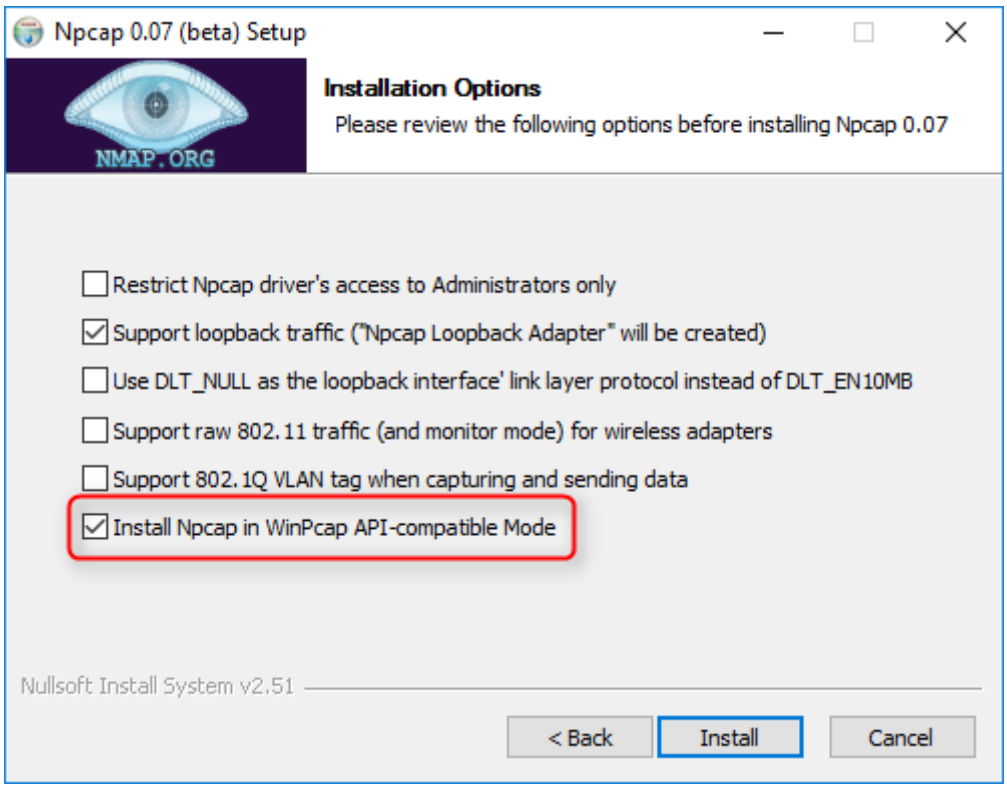
EC_T_BOOL **bFilterInput**
Filter input if EC_TRUE. This is needed on some system if the winpcap library notify the sent frames to the network adapter.

5.16.1 WinPcap, Npcap support

At least WinPcap version 4.1.2 or Npcap 0.07 r17 must be used. WinPcap version 4.1.2 is the preferred library.

The EC-Simulator installer installs WinPcap by default.

If using Npcap 0.07 r17, the WinPcap API-compatible mode must be chosen:



5.17 RDC R6040 - emllR6040

The parameters to the Real-time Ethernet Driver RDC R6040 are setup-specific. The function `CreateLinkParmsFromCmdLineR6040()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_R6040**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_R6040.

EC_T_DWORD **dwTxDmaDesCnt**
Transmit DMA descriptor buffer count. Must be a power of 2, maximum 256.

EC_T_DWORD **dwRxDmaDesCnt**
Receive DMA descriptor buffer count. Must be a power of 2, maximum 256.

5.17.1 Supported PCI devices

RDC R6040 PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_R6040** (0x17F3, 0x6040)

5.18 emllRemote

The emllRemote is used to tunnel EtherCAT® frames within a TCP socket between EC-Master and EC-Simulator.

The parameters to the Remote Driver are setup-specific. The function `CreateLinkParmsFromCmdLineRemote()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the driver instance.

struct **EC_T_LINK_PARMS_REMOTE**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_REMOTE.

EC_T_DWORD **dwSocketType**
Socket type. Must be set to 1 (emrassocktype_tcp).

EC_T_BYTE **abySrcIpAddress**[4]
Source adapter IP address (listen)

EC_T_WORD **wSrcPort**
Source port number (listen)

EC_T_BYTE **abyDstIpAddress**[4]
Destination adapter IP address (connect)

EC_T_WORD **wDstPort**
Destination port number (connect)

EC_T_BYTE **abyMac**[6]
MAC address

EC_T_DWORD **dwRxBufferCnt**
Frame buffer count for interrupt service thread (IST)

While EC-Simulator listens on the given port using emllRemote, the EC-Master can connect to it using emllRemote.

EcSimulatorHilDemo command line example:

```
$ EcSimulatorHilDemo -remote 1 0 0.0.0.0 10001 0.0.0.0 0 -f eni.xml
```

EcMasterDemo command line example:

```
$ EcMasterDemo -remote 1 1 0.0.0.0 0 127.0.0.1 10001
```

5.19 Realtek RTL8169 - emllRTL8169

The parameters to the Real-time Ethernet Driver Realtek RTL8169 are setup-specific. The function `CreateLinkParmsFromCmdLineRTL8169()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_RTL8169**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_RTL8169.

EC_T_BOOL **bNotUseDmaBuffers**
EC_FALSE: Use buffers from DMA for receive (default), EC_TRUE: use buffers from heap for receive. EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC_TRUE.

EC_T_BOOL **bAckErrInIrq**
Acknowledge errors in interrupt handler

EC_T_DWORD **dwRxBuffers**
Receive buffer count. Must be a power of 2, maximum 1024.

EC_T_DWORD **dwTxBuffers**
Transmit buffer count. Must be a power of 2, maximum 1024.

EC_T_BOOL **bNoPhyAccess**
Don't use MDIO to set up the PHY

5.19.1 RTL8169 usage under Linux

Because the Linux Kernel module de-initializes the PHY on unloading, Linux must be prevented from loading the r8169 module on startup.

5.19.2 Supported PCI devices

RealTek RTL8169 PCI specific definitions (VendorId, DeviceId)

- **PCI_DEVICE_RTL8169** (0x10EC, 0x8169)
- **PCI_DEVICE_RTL8168** (0x10EC/0x19EC, 0x8168)
- **PCI_DEVICE_RTL8169_SC** (0x10EC, 0x8167)
- **PCI_DEVICE_DLINK_RTL8169** (0x1186, 0x4300)
- **PCI_DEVICE_RTL8103** (0x10EC, 0x8136)
- **PCI_DEVICE_KILLER_E2600** (0x10EC, 0x2600)
- **PCI_DEVICE_RTL8125** (0x10EC, 0x8125)
- **PCI_DEVICE_RTL8126** (0x10EC, 0x8126)
- **PCI_DEVICE_RTL8161** (0x10EC, 0x8161)

5.20 VxWorks SNARF - emllSNARF

Using the EC-Simulator stack's SNARF Real-time Ethernet Driver it is possible to use any of the standard network drivers shipped with VxWorks. In VxWorks every network adapter is identified using a short string and a unit number in case of multiple identical network adapters. The unit numbers start with a value of 0. For example the string for the Intel Pro/100 network adapter driver is "fei". The first unit is identified using the string "fei0":

The network adapter driver has to be loaded prior to initializing the EC-Simulator stack.

Using the Real-time Ethernet Driver SNARF has some disadvantages. As the VxWorks network layering is involved in this architecture, the drivers are usually not optimized for realtime behavior and the needed CPU time is often too high to reach cycle times less than 300 to 500 microseconds. Additionally there is an impact if in parallel to EtherCAT® traffic the VxWorks application needs to use a second network card for transferring TCP/IP data. The single tNetTask is shared by all network drivers. Using a dedicated EtherCAT® driver these disadvantages can be overcome.

The parameters to the Real-time Ethernet Driver SNARF are setup-specific. The function `CreateLinkParms-FromCmdLineSNARF()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_SNARF**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_SNARF.

EC_T_CHAR **szAdapterName**[EC_SNARF_ADAPTER_NAME_SIZE]
SNARF adapter name (zero terminated)

EC_T_DWORD **dwRxBuffers**
Receive buffer count, only used in RTP context, 0: default to 20

```
#include "EcLink.h"
EC_T_LINK_PARMS_SNARF oLinkParmsAdapter;

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_SNARF));
oLinkParmsAdapter.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_SNARF;
oLinkParmsAdapter.linkParms.dwSize      = sizeof(EC_T_LINK_PARMS_SNARF);
OsStrncpy(oLinkParmsAdapter.linkParms.szDriverIdent,
        EC_LINK_PARMS_IDENT_SNARF, MAX_DRIVER_IDENT_LEN - 1);
OsStrncpy(oLinkParmsAdapter.szAdapterName, "fei0", MAX_DRIVER_IDENT_LEN - 1);
```

5.21 Linux SockRaw - emllSockRaw

emllSockRaw is part of EC-Simulator for Linux. emllSockRaw uses the network interface of the native driver, e.g., eth0, eth1, etc. The network adapter must be exclusively used for EtherCAT® and cannot be used for LAN (local area network) at the same time. Because the native Linux driver for the network adapter type is typically not fully real-time capable, it cannot be used for real time applications. If possible, an acontis Real-time Ethernet Driver, e.g. emllIntelGbe, should be used instead. emllSockRaw does not need the atemsys driver.

Note: Root privileges are required. A cycle time of 4 ms or higher may be needed.

To run the application without root privileges, set the Linux capability 'cap_net_raw' to the application.

```
$ sudo setcap 'cap_net_raw+pe' ./EcMasterDemo
```

To run python scripts without root privileges, create a Python environment and set the Linux capability 'cap_net_raw' to the python interpreter.

```
$ cd Bin/Linux
$ python3 -m venv --copies PyEnv/
$ source PyEnv/bin/activate
$ sudo setcap 'cap_net_raw+pe' PyEnv/bin/python3
```

The parameters to emllSockRaw are setup-specific. The function `CreateLinkParmsFromCmdLineSockRaw()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the parameters.

struct **EC_T_LINK_PARMS_SOCKRAW**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_SOCKRAW.

EC_T_CHAR **szAdapterName**[EC_SOCKRAW_ADAPTER_NAME_SIZE]
Native ETH device name, e.g. "eth0" (zero terminated)

EC_T_BOOL **bDisableForceBroadcast**
Don't change target MAC address to FF:FF:FF:FF:FF:FF

EC_T_BOOL **bReplacePaddingWithNopCmd**
Prevent adding Ethernet padding to work-around EtherCAT corruption bugs from native Linux driver(s)

EC_T_BOOL **bUsePacketMmapRx**
Use PACKET_MMAP PACKET_RX_RING for receive

EC_T_BOOL **bSetCoalescingParms**
Set Coalescing parameters to enhance the link layer performance

EC_T_BOOL **bSetPromiscuousMode**
Enable promiscuous mode at network adapter

5.22 Linux SockXdp - emllSockXdp

The Real-time Ethernet Driver SockXdp does not need the atemsys driver and uses already established Ethernet adapters, e.g. eth0, eth1, etc. It is strongly recommended to use a separate network adapter to connect EtherCAT® network adapters. If the main network adapter is used for both EtherCAT® network adapters and the local area network there may be a major impact on the local area network operation.

Note: Root privileges are required.

The parameters to the Real-time Ethernet Driver SockXdp are setup-specific. The function `CreateLinkParmsFromCmdLineSockXdp()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_SOCKXDP**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_SOCKXDP.

EC_T_CHAR **szAdapterName**[EC_SOCKXDP_ADAPTER_NAME_SIZE]
Native ETH device name, e.g. “eth0” (zero terminated)

EC_T_WORD **wQueueId**
Used NIC Queue Id

EC_T_XDP_MODE **eXdpMode**
XDP mode

EC_T_BOOL **bDisableForceBroadcast**
Don’t change target MAC address to FF:FF:FF:FF:FF:FF

EC_T_BOOL **bReplacePaddingWithNopCmd**
Prevent adding Ethernet padding to work-around EtherCAT corruption bugs from native Linux driver(s)

enum **EC_T_XDP_MODE**
Values:

enumerator **eXDP_SKB_MODE**
Generic XDP, XDP runs in the standard network stack. This is the fallback mode if native XDP is not supported

enumerator **eXDP_DRV_MODE**
Native XDP, Runs XDP directly in the NIC driver. This is the recommended mode for the best performance

enumerator **eXDP_HW_MODE**
Offloaded XDP, XDP programs are offloaded to the network card itself, running on the NIC firmware, not tested

enumerator **eXDP_DRV_ZEROCOPY**
Native XDP with ZEROCOPY

5.22.1 Linux System Requirements

To use the XDP Real-time Ethernet Driver the following system requirements need to be met:

- **Your kernel have to support XDP, check if the following entry is activated in your `.config` file.**
  `CONFIG_XDP_SOCKETS=y`

  If it is not activated you need to rebuild your kernel with `CONFIG_XDP_SOCKETS`.

5.22.2 Getting started

- Download libxdp Version 1.4.3 from https://github.com/xdp-project/xdp-tools and install it.
- Download libbpf Version 1.5.0 from https://github.com/libbpf/libbpf and install it.
- Update the linux drivers of the NIC that would be used
- **If the NIC supports XDP, turn off xdp generic mode:**

```
$ ip link set dev eth0 xdpgeneric off
```

- **If the NIC have multiple queue support, use one queue instead of multiple queues:**

```
$ ethtool -L eth0 combined 1
```

5.23 Windows TAP - emllTap

The parameters to the Windows TAP Real-time Ethernet Driver are setup-specific. The function `CreateLinkParmsFromCmdLineTap()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

struct **EC_T_LINK_PARMS_TAP**

Public Members

EC_T_LINK_PARMS **linkParms**
Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_TAP.

EC_T_CHAR **szAdapterName**[EC_TAP_ADAPTER_NAME_SIZE]
Native tap device name, e.g. “tap0” (zero terminated)

EC_T_CHAR **szAdapterGuid**[EC_TAP_ADAPTER_GUID_SIZE]
GUID of virtual interface to connect (zero terminated)

EC_T_BYTE **abyMac**[6]
MAC address

EC_T_DWORD **dwRxBufferCnt**
Frame buffer count for IST

The adapter must be manually installed as described in the OpenVPN’s Windows TAP-driver manual, see https://community.openvpn.net/openvpn/wiki/GettingTapWindows .

While EC-Simulator is connected to the adapter using emllTap, the MainDevice can use it with e.g. emllNdis.

6 Application programming interface, reference

6.1 Header files

Function prototypes, definitions etc. of the API can be found in the header file `EcSimulator.h` which is the main header file to include when using EC-Simulator.

6.2 Generic API return status values

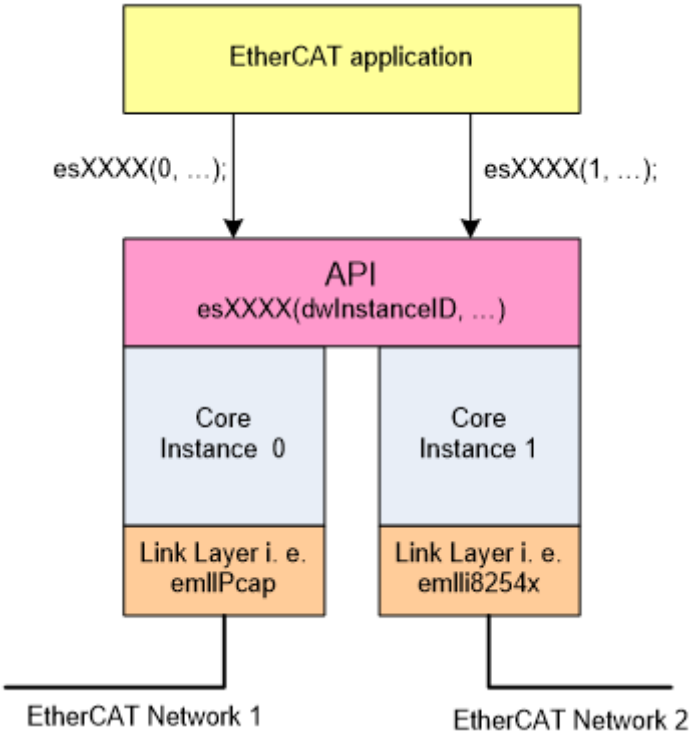
Most of the functions and also some notifications will return an error status value to indicate whether a function was successfully executed or not. Some of the return status values have a generic meaning unspecific to the called API function.

- *EC_E_NOERROR*: The function was successfully executed.
- *EC_E_NOTSUPPORTED*: Unsupported feature or functionality.
- *EC_E_BUSY*: The stack is currently busy and the function has to be re-tried at a later time.
- *EC_E_NOMEMORY*: Not enough memory or frame buffer resources available.
- *EC_E_INVALIDPARM*: Invalid or inconsistent parameters.
- *EC_E_TIMEOUT*: Timeout error.
- *EC_E_SLAVE_ERROR*: A SubDevice error was detected.
- *EC_E_INVALID_SLAVE_STATE*: The SubDevice is not in the requested state to execute the operation (e.g. when initiating a mailbox transfer the SubDevice must be at least in PREOP state).
- *EC_E_SLAVE_NOT_ADDRESSABLE*: The SubDevice does not respond to its station address (e.g. when requesting its AL_STATUS value). The SubDevice may be removed from the bus or powered-off.
- *EC_E_LINK_DISCONNECTED*: Link cable not connected.
- *EC_E_MASTERCORE_INACCESSIBLE*: MainDevice core inaccessible. This result code usually means a remote connected server / EC-Simulator stack does not respond anymore.

6.3 Multiple EtherCAT® Network Support

6.3.1 Overview

The acontis EC-Simulator stack supports multiple EtherCAT® network instances within the same application process. The first parameter of all esXxx API functions is the Instance ID, starting with 0. The maximum Instance ID of the EC-Simulator SDK is defined as MAX_NUMOF_SIMULATOR_INTERFACES - 1. Implementing multiple EtherCAT® network instances is realized by using multiple Instance IDs.



The Real-time Ethernet Driver can be of same or different type (emllPcap, emllIntelGbe, …).

6.3.2 Licensing

For each network identified with the unique Instance ID a separate runtime license is required.

6.4 General functions

6.4.1 esInitSimulator

EC_T_DWORD **esInitSimulator** (
EC_T_DWORD dwInstanceId,
const *EC_T_SIMULATOR_INIT_PARMS* *pParms
)

Initialize EC-Simulator HiL.

This function has to be called prior to calling any other function of EC-Simulator HiL. For EC-Simulator SiL (emllSimulator) this function is implicitly called by emInitMaster() and may not be called explicitly.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **pParms** – [in] Pointer to init parameters

Returns

EC_E_NOERROR or error code

struct **EC_T_SIMULATOR_INIT_PARMS**

Public Members

EC_T_DWORD **dwSignature**
[in] Set to SIMULATOR_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(EC_T_SIMULATOR_INIT_PARMS)

EC_T_DWORD **dwSimulatorAddress**
[in] Reserved

EC_T_OS_PARMS ***pOsParms**
[in] OS layer parameters

EC_T_LINK_PARMS ***apLinkParms**[EC_SIMULATOR_MAX_LINK_PARMS]
[in] Link layer parameters

EC_T_DWORD **dwBusCycleTimeNsec**
[in] Bus cycle time in nanoseconds [ns]

EC_T_BOOL **bDisableProcessDataImage**
[in] Disable Process Data Image (Process Data Variables not supported)

EC_T_SIMULATOR_DEVICE_CONNECTION_DESC
aoDeviceConnection[EC_SIMULATOR_MAX_LINK_PARMS]
[in] see EC_SIMULATOR_DEVICE_CONNECTION_TYPE…

EC_T_BOOL **bConnectHcGroups**
[in] Connect hot connect groups in topology (floating group heads to free ports)

EC_T_PERF_MEAS_INTERNAL_PARMS **PerfMeasInternalParms**
[in] Internal performance measurement parameters

EC_T_BOOL **bApiLockByApp**
[in] Lock pending API against esDeinitSimulator(). EC_FALSE (default): locked internally. EC_TRUE: application is responsible for locking.

struct **EC_T_OS_PARMS**

Public Members

EC_T_DWORD **dwSignature**
[in] Set to EC_OS_PARMS_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(EC_T_OS_PARMS)

EC_T_LOG_PARMS ***pLogParms**
[in] Pointer to logging parameters

EC_PF_SYSTIME **pfSystemTimeGet**
[in] Function to get host time in nanoseconds since 1st January 2000. Used as time base for DC Initialization.

EC_T_DWORD **dwSupportedFeatures**
[in/out] Reserved

EC_PF_QUERY_MSEC_COUNT **pfSystemQueryMsecCount**
[in] Function to get system's ms count

struct **EC_T_LOG_PARMS**

Public Members

EC_T_DWORD **dwLogLevel**
[in] Log level. See *EC_LOG_LEVEL…*

EC_PF_LOGMSGHK **pfLogMsg**
[in] Log callback function called on every message

EC_T_LOG_CONTEXT ***pLogContext**
[in] Log context to be passed to log callback

EC_LOG_LEVEL… The following log levels are defined:

EC_LOG_LEVEL_SILENT

EC_LOG_LEVEL_ANY

EC_LOG_LEVEL_CRITICAL

EC_LOG_LEVEL_ERROR

EC_LOG_LEVEL_WARNING

EC_LOG_LEVEL_INFO

EC_LOG_LEVEL_INFO_API

EC_LOG_LEVEL_VERBOSE

EC_LOG_LEVEL_VERBOSE_ACYC

EC_LOG_LEVEL_VERBOSE_CYC

EC_LOG_LEVEL_UNDEFINED

typedef EC_T_DWORD (***EC_PF_LOGMSGHK**)(EC_T_LOG_CONTEXT *pContext, EC_T_DWORD dwLogMsgSeverity, const EC_T_CHAR *szFormat, ...)

Param pContext
[in] Context pointer. This pointer is used as parameter when the callback function is called.

Param dwLogMsgSeverity
[in] Log message severity, EC_LOG_LEVEL_…

Param szFormat
[in] String that contains the text to be written. It can optionally contain embedded format specifiers that are replaced by the values specified in subsequent additional arguments and formatted as requested.

Return
EC_E_NOERROR or error code

Log messages are passed from the EC-Simulator to the callback given at *EC_T_LOG_PARMS::pfLogMsg*. The example program demonstrates how messages can be processed by the application, see `Examples/Common/EcLogging.cpp`. For performance reasons the EC-Simulator automatically filters log messages according to *EC_T_LOG_PARMS::dwLogLevel*. E.g. messages of severity *EC_LOG_LEVEL_WARNING* are not passed to the application if *EC_T_LOG_PARMS::dwLogLevel* is set to *EC_LOG_LEVEL_ERROR*.

The application can provide customized log message handlers of type *EC_PF_LOGMSGHK* if the default handler in `EcLogging.cpp` does not fulfill the application's needs. Note: The callback is typically called from the Job Task's context and should return as fast as possible.

struct **EC_T_PERF_MEAS_INTERNAL_PARMS**

Public Members

EC_T_BOOL **bEnabled**
[in] Enable/disable internal performance counters

EC_T_PERF_MEAS_COUNTER_PARMS **CounterParms**
[in] Timer function settings. When not provided OsMeasGetCounterTicks is used.

EC_T_PERF_MEAS_HISTOGRAM_PARMS **HistogramParms**
[in] Histogram settings. When not provided the histogram is disabled.

struct **EC_T_PERF_MEAS_COUNTER_PARMS**

Public Members

EC_PF_PERF_MEAS_GETCOUNTERTICKS **pfGetCounterTicks**
[in] Function returning the current counter ticks

EC_T_VOID ***pvGetCounterTicksContext**
[in] Context passed into GetCounterTicks

EC_T_UINT64 **qwFrequency**
[in] Frequency in Hz used by the timer in GetCounterTicks

typedef EC_T_UINT64 (***EC_PF_PERF_MEAS_GETCOUNTERTICKS**)(EC_T_VOID *pvContext)

Param pvContext
[in] Arbitrarily application-defined parameter passed to callback

struct **EC_T_PERF_MEAS_HISTOGRAM_PARMS**

Public Members

EC_T_DWORD **dwBinCount**
[in] Amount of bins to use for the histogram

EC_T_UINT64 **qwMinTicks**
[in] Results below qwMinTicks are stored in the first bin

EC_T_UINT64 **qwMaxTicks**
[in] Results above qwMaxTicks are stored in the last bin

6.4.2 esDeinitSimulator

EC_T_DWORD **esDeinitSimulator** (EC_T_DWORD dwInstanceId)
Deinitialize EC-Simulator HiL.

Waits for pending API calls if *esInitSimulator()* was called with *EC_T_SIMULATOR_INIT_PARMS::bApiLockByApp* = EC_FALSE (default). For EC-Simulator SiL (emllSimulator) this function is implicitly called by emDeinitMaster() and may not be called explicitly.

–

Parameters
dwInstanceId – [in] Simulator Instance ID

Returns
EC_E_NOERROR or error code

6.4.3 esGetSimulatorParms

EC_T_DWORD **esGetSimulatorParms** (
EC_T_DWORD dwInstanceId,
EC_T_SIMULATOR_INIT_PARMS *pParms,
EC_T_DWORD dwParmsBufSize
)

Get Simulator parameters provided by esInitSimulator(…) or esSetSimulatorParms(…).

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **pParms** – [out] Pointer to simulator parameters
- **dwParmsBufSize** – [in] Size of buffer at pParms

Returns
EC_E_NOERROR or error code

See also:

esInitSimulator()

6.4.4 esSetSimulatorParms

EC_T_DWORD **esSetSimulatorParms** (
EC_T_DWORD dwInstanceId,
const *EC_T_SIMULATOR_INIT_PARMS* *pParms
)

Change Simulator parameters provided by esInitSimulator(…).

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **pParms** – [in] Pointer to simulator parameters

Returns
EC_E_NOERROR or error code

See also:

esInitSimulator()

6.4.5 esSetLogParms

EC_T_DWORD **esSetLogParms** (EC_T_DWORD dwInstanceID, *EC_T_LOG_PARMS* *pLogParms)

Sets log parameters. Used to change the parameters provided by esInitSimulator().

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pLogParms** – [in] New Log parameters

See also:

esInitSimulator()

See also:

esSetSimulatorParms()

6.4.6 esGetSrcMacAddress

EC_T_DWORD **esGetSrcMacAddress** (
EC_T_DWORD dwInstanceID,
ETHERNET_ADDRESS *pMacSrc
)

Gets the source MAC address.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMacSrc** – [out] 6-byte buffer to write source MAC address to

Returns
EC_E_NOERROR or error code

Refers to adapter from EC_T_SIMULATOR_INIT_PARMS.pLinkParms at *esInitSimulator()*.

6.4.7 esSetLicenseKey (HiL)

EC_T_DWORD **esSetLicenseKey** (
EC_T_DWORD dwInstanceID,
const EC_T_CHAR *szLicenseKey
)

Sets the license key for the protected version of EC-Master.

Must be called after initialization and before configuration. This function may not be called if a non protected version is used.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szLicenseKey** – [in] License key as zero terminated string with 26, 53 or 56 characters

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range
- *EC_E_INVALIDSIZE* if the format of the license key is wrong. The correct length is 26, 53 or 56 characters.
- *EC_E_LICENSE_MISSING* if the license key doesn't match the MAC Address

Must be called after *esInitSimulator()* and before *esConfigureNetwork()*. This function may not be called if a non protected version is used.

Example:

```
dwRes = esSetLicenseKey(dwInstanceID, "DA1099F2-15C249E9-54327FBC");
```

6.4.8 esSetOemKey (HiL)

static EC_T_DWORD **esSetOemKey** (EC_T_DWORD dwInstanceID, EC_T_UINT64 qwOemKey)

Must be called after *esInitSimulator()* and before *esConfigureNetwork()*.

Example:

```
dwRes = esSetOemKey(dwInstanceID, 0x1234567812345678);
```

6.4.9 esConfigureNetwork

EC_T_DWORD **esConfigureNetwork** (
    EC_T_DWORD dwInstanceID,
    EC_T_CNF_TYPE eCnfType,
    EC_T_PBYTE pbyCnfData,
    EC_T_DWORD dwCnfDataLen
)

Configure the Network.

This function must be called after the initialization. Among others the EtherCAT topology defined in the given XML configuration file will be stored internally.

Analyzing the network including mailbox communication can be done without specifying an ENI file using eCnfType_GenPreopENI.

Note: A client must not be registered prior to calling this function. Existing client registrations will be dropped.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eCnfType** – [in] Type of configuration data provided
- **pbyCnfData** – [in] Filename / configuration data, or EC_NULL if eCnfType is eCnfType_GenPreopENI
- **dwCnfDataLen** – [in] Length of configuration data in byte, or zero if eCnfType is eCnfType_GenPreopENI

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized or eCnfType is eCnfType_GenPreopENI or eCnfType_GenOpENI and link is disconnected
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or pParms is EC_NULL contains some values out of range
- *EC_E_LINK_DISCONNECTED* if link is disconnected
- *EC_E_FEATURE_DISABLED* if a configured feature is not included in the license key
- *EC_E_NOTSUPPORTED* if a configured feature is not supported (e.g not compiled in the library)

- *EC_E_CFGFILENOTFOUND* if the ENI file cannot be found
- *EC_E_WRONG_FORMAT* if format errors have been detected in the ENI or EEPROM in case of eCnfType_GenPreopENI or eCnfType_GenOpENI
- *EC_E_OEM_SIGNATURE_MISMATCH* if the OEM signature in the ENI file doesn't match the used OEM key
- *EC_E_ENI_ENCRYPTION_WRONG_VERSION* if the ENI encryption version is not supported (e.g. the library is too old)
- *EC_E_ENI_ENCRYPTED* if the ENI is encrypted and no OEM key has been set
- *EC_E_XML_CYCCMDS_MISSING* if the ENI doesn't contain cyclic commands
- *EC_E_XML_ALSTATUS_READ_MISSING* if the ENI doesn't contain any read AL status command
- *EC_E_XML_CYCCMDS_SIZEMISMATCH* if the size of the cyclic commands in the ENI mismatch
- *EC_E_XML_INVALID_INP_OFF* if some input offsets in the ENI are invalid
- *EC_E_XML_INVALID_OUT_OFF* if some output offsets in the ENI are invalid
- *EC_E_XML_INVALID_CMD_WITH_RED* if the ENI contains LRW commands and cable redundancy is configured
- *EC_E_XML_PREV_PORT_MISSING* if some previous port information are missing in the ENI
- *EC_E_XML_DC_CYCCMDS_MISSING* if the DC related cyclic commands are missing in the ENI
- *EC_E_XML_AOE_NETID_INVALID* if the ENI contains some invalid NetID

enum **EC_T_CNF_TYPE**

Values:

enumerator **eCnfType_Unknown**

enumerator **eCnfType_Filename**

pbyCnfData: ENI filename to read

enumerator **eCnfType_Data**

pbyCnfData: ENI data

enumerator **eCnfType_Datadiag**

pbyCnfData: ENI data for diagnosis

enumerator **eCnfType_GenPreopENI**

Generate ENI based on bus-scan result to get into PREOP state

enumerator **eCnfType_GenPreopENIWithCRC**

Same as eCnfType_GenPreopENI with CRC protection

enumerator **eCnfType_GenOpENI**

Generate ENI based on bus-scan result to get into OP state. The default PDO mapping read from the slaves is activated. See ETG2010 "SII Specification", Table 14 "Structure Category TXPDO and RXPDO for each PDO".

enumerator **eCnfType_None**
Reset configuration

enumerator **eCnfType_ConfigData**
pbyCnfData: Binary structured configuration

enumerator **eCnfType_GenOpENINoStrings**
Generate ENI based on bus-scan result to get into OP state, does not read strings from EEPROM

enumerator **eCnfType_FileByApp**
File access provided by user application, See EC_T_CNF_FILEBYAPP_DESC

enumerator **eCnfType_GenEBI**
Generate EBI based on bus-scan result

6.4.10 esGetCfgSlaveInfo

EC_T_DWORD **esGetCfgSlaveInfo** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_CFG_SLAVE_INFO *pSlaveInfo
)
Return information about a configured slave from the ENI file.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveInfo** – [out] Information about the slave

Returns
EC_E_NOERROR or error code

Content of *EC_T_CFG_SLAVE_INFO* is subject to be extended.

struct **EC_T_CFG_SLAVE_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] Slave's ID to bind bus slave and config slave information

EC_T_CHAR **abyDeviceName**[ECAT_DEVICE_NAMESIZE]
[out] Slave's configured name (80 Byte) (from ENI file)

EC_T_DWORD **dwHCGroupIdx**
[out] Index of Hot Connect group, 0 for mandatory

EC_T_BOOL **bIsPresent**
[out] Slave present on bus

EC_T_BOOL **bIsHCGroupPresent**
[out] Slave's Hot Connect group present on bus

EC_T_DWORD **dwVendorId**
[out] Vendor identification (from ENI file)

EC_T_DWORD **dwProductCode**
[out] Product code (from ENI file)

EC_T_DWORD **dwRevisionNumber**
[out] Revision number (from ENI file)

EC_T_DWORD **dwSerialNumber**
[out] Serial number (from ENI file)

EC_T_WORD **wStationAddress**
[out] Slave's configured station address (from ENI file)

EC_T_WORD **wAutoIncAddress**
[out] Slave's auto increment address (may differ from ENI file)

EC_T_DWORD **dwPdOffsIn**
[out] Process input data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeIn**
[out] Process input data bit size (from ENI file)

EC_T_DWORD **dwPdOffsOut**
[out] Process output data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeOut**
[out] Process output data bit size (from ENI file)

EC_T_DWORD **dwPdOffsIn2**
[out] 2nd sync unit process input data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeIn2**
[out] 2nd sync unit process input data bit size (from ENI file)

EC_T_DWORD **dwPdOffsOut2**
[out] 2nd sync unit process output data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeOut2**
[out] 2nd sync unit process output data bit size (from ENI file)

EC_T_DWORD **dwPdOffsIn3**
[out] 3rd sync unit process input data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeIn3**
[out] 3rd sync unit process input data bit size (from ENI file)

EC_T_DWORD **dwPdOffsOut3**
[out] 3rd sync unit process output data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeOut3**
[out] 3rd sync unit process output data bit size (from ENI file)

EC_T_DWORD **dwPdOffsIn4**
[out] 4th sync unit process input data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeIn4**
[out] 4th sync unit process input data bit size (from ENI file)

EC_T_DWORD **dwPdOffsOut4**
[out] 4th sync unit process output data bit offset (from ENI file)

EC_T_DWORD **dwPdSizeOut4**
[out] 4th sync unit process output data bit size (from ENI file)

EC_T_DWORD **dwMbxSupportedProtocols**
[out] Mailbox protocols supported by the slave (from ENI file). Combination of *Supported mailbox protocols* flags.

EC_T_DWORD **dwMbxOutSize**
[out] Mailbox output byte size (from ENI file)

EC_T_DWORD **dwMbxInSize**
[out] Mailbox input byte size (from ENI file)

EC_T_DWORD **dwMbxOutSize2**
[out] Bootstrap mailbox output byte size (from ENI file)

EC_T_DWORD **dwMbxInSize2**
[out] Bootstrap mailbox input byte size (from ENI file)

EC_T_BOOL **bDcSupport**
[out] Slave supports DC (from ENI file)

EC_T_WORD **wNumProcessVarsInp**
[out] Number of input process data variables (from ENI file)

EC_T_WORD **wNumProcessVarsOutp**
[out] Number of output process data variables (from ENI file)

EC_T_WORD **wPrevStationAddress**
[out] Station address of the previous slave (from ENI file)

EC_T_WORD **wPrevPort**
[out] Connected port of the previous slave (from ENI file)

EC_T_WORD **wIdentifyAdo**
[out] ADO used for identification command (from ENI file)

EC_T_WORD **wIdentifyData**
[out] Identification value to be validated (from ENI file)

EC_T_BYTE **byPortDescriptor**
[out] Port descriptor (ESC register 0x0007) (from ENI file)

EC_T_WORD **wWkcStateDiagOffsIn**[EC_CFG_SLAVE_PD_SECTIONS]
[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Recv[1..4]/BitStart): 0xFFFFFFFF = offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid.

EC_T_WORD **wWkcStateDiagOffsOut**[EC_CFG_SLAVE_PD_SECTIONS]
[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Send[1..4]/BitStart): 0xFFFFFFFF = offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid.

EC_T_WORD **awMasterSyncUnitIn**[EC_CFG_SLAVE_PD_SECTIONS]
[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

EC_T_WORD **awMasterSyncUnitOut**[EC_CFG_SLAVE_PD_SECTIONS]
[out] Sync Unit (ENI: ProcessData/RxPdo[1..4]@Su)

EC_T_BOOL **bDisabled**
[out] Slave disabled by API SetSlaveDisabled / SetSlavesDisabled.

EC_T_BOOL **bDisconnected**
[out] Slave disconnected by API SetSlaveDisconnected / SetSlavesDisconnected.

EC_T_BOOL **bExtended**
[out] Slave generated by API ConfigExtend

EC_T_BOOL **bDcReferenceClock**
[out] Slave is reference clock (from ENI file)

EC_T_BOOL **bDcPotentialRefClock**
[out] Slave can be used as a reference clock (from ENI file)

EC_T_DWORD **dwDcCycleTime0**
[out] Cycle time of Sync0 event [ns] (from ENI file)

EC_T_DWORD **dwDcCycleTime1**
[out] Calculated value dwDcCycleTime1 [ns] = Cycle time of Sync1 event - Cycle time of Sync1 event + Shift time of Sync0 event (from ENI file)

EC_T_INT **nDcShiftTime**
[out] Shift time of Sync0 event [ns] (from ENI file)

Supported mailbox protocols flags

EC_MBX_PROTOCOL_AOE

EC_MBX_PROTOCOL_EOE

EC_MBX_PROTOCOL_COE

EC_MBX_PROTOCOL_FOE

EC_MBX_PROTOCOL_SOE

EC_MBX_PROTOCOL_VOE

esGetCfgSlaveInfo() Example

```
/* get information about slave configured in ENI file */
EC_T_CFG_SLAVE_INFO oSlaveInfo;
OsMemset(&oSlaveInfo, 0, sizeof(EC_T_CFG_SLAVE_INFO));
dwRes = esGetCfgSlaveInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveInfo);
```

6.4.11 esGetCfgSlaveSmInfo

EC_T_DWORD **esGetCfgSlaveSmInfo** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_CFG_SLAVE_SM_INFO *pSlaveSmInfo
)

Return SyncManager information of a configured slave from the ENI file.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveSmInfo** – [out] Information about the slave.

Returns

EC_E_NOERROR or error code

struct **EC_T_CFG_SLAVE_SM_ENTRY**

Public Members

EC_T_WORD **wPhysAddr**
[out] ESC (0x800 + y * 8)

EC_T_WORD **wLength**
[out] ESC (0x802 + y * 8)

EC_T_BYTE **byOpMode**
[out] Bits 0..1 ESC (0x804 + y * 8)

EC_T_BYTE **byDirection**
[out] Bits 2..3 ESC (0x804 + y * 8)

EC_T_DWORD **dwPdBitOffs**
[out] Process input data bit offset (from ENI file)

EC_T_DWORD **dwPdBitSize**
[out] Process input data bit size (from ENI file)

EC_T_WORD **wWkcStateDiagBitOffs**
[out] Offset of WkcState bit in diagnosis image

EC_T_WORD **wMasterSyncUnit**
[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

struct **EC_T_CFG_SLAVE_SM_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] Slave ID

EC_T_DWORD **dwSmInfoNumOf**
[out] Number of available sync managers

EC_T_CFG_SLAVE_SM_ENTRY **aoSmInfos**[ECREG_SYNCMANAGER_MAX_NUMOF]
[out] Sync managers info

esGetCfgSlaveSmInfo() Example

```
/* get information about slave's sync managers configured in ENI file */
EC_T_CFG_SLAVE_SM_INFO oSlaveSmInfo;
OsMemset(&oSlaveSmInfo, 0, sizeof(EC_T_CFG_SLAVE_SM_INFO));
dwRes = esGetCfgSlaveSmInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveSmInfo);
```

6.4.12 esSetSlaveSscApplication

EC_T_DWORD **esSetSlaveSscApplication** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_SLAVE_SSC_APPL_DESC *pApp
)
Register slave application's callback functions for slave without device emulation.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **pApp** – [in] Pointer to application descriptor

Returns
EC_E_NOERROR or error code

struct **EC_T_SLAVE_SSC_APPL_DESC**

Public Members

EC_T_DWORD **dwSignature**
[in] Set to SIMULATOR_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(EC_T_SLAVE_SSC_APPL_DESC)

EC_T_VOID ***pvContext**
[in] First parameter of SSC_APPL-callbacks

EC_PF_SSC_APPL_MAIN_INIT **pfnMainInit**
[in] SSC_APPL-callback for APPL_MainInit, called by SSC when slave initialization finished

EC_PF_SSC_APPL_ACK_ERROR_IND **pfnAckErrorInd**
[in] SSC_APPL-callback for APPL_AckErrorInd, called by SSC on error acknowledge by master

EC_PF_SSC_APPL_APPLICATION **pfnApplication**
[in] SSC_APPL-callback for APPL_Application, called after frame processing

EC_PF_SSC_APPL_GET_DEVICE_ID **pfnGetDeviceID**
[in] SSC_APPL-callback for APPL_GetDeviceID, called by SSC on Explicit Device ID request by master

EC_PF_SSC_APPL_START_MAILBOX_HANDLER **pfnStartMailboxHandler**
[in] SSC_APPL-callback for APPL_StartMailboxHandler, called by SSC on INIT to PREOP and INIT to BOOT transition

EC_PF_SSC_APPL_STOP_MAILBOX_HANDLER **pfnStopMailboxHandler**
[in] SSC_APPL-callback for APPL_StopMailboxHandler, called by SSC on PREOP to INIT and BOOT to INIT transition

EC_PF_SSC_APPL_START_INPUT_HANDLER **pfnStartInputHandler**
[in] SSC_APPL-callback for APPL_StartInputHandler, called by SSC on PREOP to SAFEOP transition (even if no INPUTs configured)

EC_PF_SSC_APPL_STOP_INPUT_HANDLER **pfnStopInputHandler**
[in] SSC_APPL-callback for APPL_StopInputHandler, called by SSC on SAFEOP to PREOP transition (even if no INPUTs configured)

EC_PF_SSC_APPL_START_OUTPUT_HANDLER **pfnStartOutputHandler**
[in] SSC_APPL-callback for APPL_StartOutputHandler, called by SSC on SAFEOP to OP transition (even if no OUTPUTs configured)

EC_PF_SSC_APPL_STOP_OUTPUT_HANDLER **pfnStopOutputHandler**
[in] SSC_APPL-callback for APPL_StopOutputHandler, called by SSC on OP to SAFEOP transition (even if no OUTPUTs configured)

EC_PF_SSC_APPL_GENERATE_MAPPING **pfnGenerateMapping**
[in] SSC_APPL-callback for APPL_GenerateMapping, called by SSC on PREOP to SAFEOP transition

EC_PF_SSC_APPL_INPUT_MAPPING **pfnInputMapping**
: [in] SSC_APPL-callback for APPL_InputMapping, called by SSC to map (copy) INPUTs

EC_PF_SSC_APPL_OUTPUT_MAPPING **pfnOutputMapping**
: [in] SSC_APPL-callback for APPL_OutputMapping, called by SSC to map (copy) OUTPUTs

EC_T_PF_AOE_READ_CB **pfnAoeRead**
: [in] SSC_APPL-callback for APPL_AoeRead, called by SSC on AoE read request

EC_T_PF_AOE_WRITE_CB **pfnAoeWrite**
: [in] SSC_APPL-callback for APPL_AoeWrite, called by SSC on AoE write request

EC_T_PF_AOE_READWRITE_CB **pfnAoeReadWrite**
: [in] SSC_APPL-callback for APPL_AoeReadWrite, called by SSC on AoE read/write request

EC_PF_SSC_APPL_EOE_RECEIVE **pfnEoeReceive**
: [in] SSC_APPL-callback for APPL_EoeReceive, called by SSC on received EoE Ethernet frame completion

EC_PF_SSC_APPL_EOE_SETTING_IND **pfnEoeSettingInd**
: [in] SSC_APPL-callback for APPL_EoeSettingInd, called by SSC on EoE settings received

EC_PF_SSC_APPL_FOE_WRITE **pfnFoeWrite**
: [in] SSC_APPL-callback for APPL_FoeWrite, called by SSC on FoE write request

EC_PF_SSC_APPL_FOE_WRITE_DATA **pfnFoeWriteData**
: [in] SSC_APPL-callback for APPL_FoeWriteData, called by SSC when FoE data received

EC_PF_SSC_APPL_FOE_READ **pfnFoeRead**
: [in] SSC_APPL-callback for APPL_FoeRead, called by SSC on FoE read request to transmit first FoE data

EC_PF_SSC_APPL_FOE_READ_DATA **pfnFoeReadData**
: [in] SSC_APPL-callback for APPL_FoeReadData, called by SSC to transmit next FoE read data, 2..n

EC_PF_SSC_APPL_FOE_ERROR **pfnFoeError**
: [in] SSC_APPL-callback for APPL_FoeError, called by SSC on FoE abort by master

EC_PF_SSC_APPL_SET_PROC_VAR_PTR **pfnSetProcVarPtr**
: [in] SSC_APPL-callback for APPL_SetProcVarPtr, called on *esConfigureNetwork()* for each process data variable

See also:

- *ADS over EtherCAT® (AoE)*
- *Ethernet over EtherCAT® (EoE)*
- *File access over EtherCAT® (FoE)*
- `esInitSimulator()`
- `esConfigureNetwork()`

6.4.13 esRegisterClient

EC_T_DWORD **esRegisterClient** (
EC_T_DWORD dwInstanceID,
EC_PF_NOTIFY pfnNotify,
EC_T_VOID *pCallerData,
EC_T_REGISTERRESULTS *pRegResults
)

Registers a client on the EC-Master.

It must be called after configuration, otherwise the registration handle is lost. This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pfnNotify** – [in] Notification callback function. This function will be called every time a state change occurs, an error occurs or a mailbox transfer terminates.
- **pCallerData** – [in] Pointer to a caller data area which will be passed to the client on every notification callback
- **pRegResults** – [out] Registration results, a pointer to a structure of type *EC_T_REGISTERRESULTS*

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOMEMORY* if some memory cannot be allocated

typedef EC_T_DWORD (***EC_PF_NOTIFY**)(EC_T_DWORD dwCode, *EC_T_NOTIFYPARMS* *pParms)

Param dwCode
[in] Notification code, see EC_NOTIFY_…

Param pParms
[in] Notification code depending data

struct **EC_T_REGISTERRESULTS**

Public Members

EC_T_DWORD **dwClntId**
[out] Client ID

EC_T_BYTE ***pbyPDIn**
[out] Pointer to process data input memory

EC_T_DWORD **dwPDInSize**
[out] Size of process data input memory (in bytes)

EC_T_BYTE ***pbyPDOut**
[out] Pointer to process data output memory

EC_T_DWORD **dwPDOutSize**
[out] Size of process data output memory (in bytes)

6.4.14 esUnregisterClient

EC_T_DWORD **esUnregisterClient** (EC_T_DWORD dwInstanceID, EC_T_DWORD dwClntId)
Unregister a client from the EtherCAT master.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClntId** – [in] Client ID determined when registering with the master

Returns
EC_E_NOERROR or error code

6.4.15 esExecJob

EC_T_DWORD **esExecJob** (
EC_T_DWORD dwInstanceID,
EC_T_USER_JOB eUserJob,
EC_T_USER_JOB_PARMS *pUserJobParms
)
Execute or initiate the requested job.

To achieve maximum speed, this function is implemented non re-entrant. It is highly recommended that only one single task is calling all required jobs to run the stack. If multiple tasks are calling this function, the calls have to be synchronized externally. Calling it in a context that doesn't support operating system calls can lead to unpredictable behavior.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eUserJob** – [in] User requested job
- **pUserJobParms** – [in] Optional user job parameters

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_LINK_DISCONNECTED* if the link is disconnected
- *EC_E_FEATURE_DISABLED* for eUsrJob_SwitchEoeFrames if EC_IOCTL_SET_EOE_DEFFERED_SWITCHING_ENABLED hasn't be called before
- *EC_E_ADS_IS_RUNNING* if the ADS server is running

Brief job overview:

enum **EC_T_USER_JOB**

Values:

enumerator **eUsrJob_Undefined**

enumerator **eUsrJob_ProcessAllRxFrames**

enumerator **eUsrJob_SendAllCycFrames**

enumerator **eUsrJob_MasterTimer**

enumerator **eUsrJob_SendAcycFrames**

enumerator **eUsrJob_SendCycFramesByTaskId**

enumerator **eUsrJob_MasterTimerMinimal**

enumerator **eUsrJob_ProcessRxFramesByTaskId**

enumerator **eUsrJob_ProcessAcycRxFrames**

enumerator **eUsrJob_SwitchEoeFrames**

enumerator **eUsrJob_StartTask**

enumerator **eUsrJob_StopTask**

enumerator **eUsrJob_StampSendAllCycFrames**

enumerator **eUsrJob_StampSendCycFramesByTaskId**

enumerator **eUsrJob_SimulatorTimer**

enumerator **eUsrJob_MonitorTimer**

union **EC_T_USER_JOB_PARMS**

Public Members

EC_T_BOOL **bAllCycFramesProcessed**

EC_T_DWORD **dwNumFramesSent**

EC_T_DWORD **dwTaskIdToSend**

struct *EC_T_USER_JOB_PARMS*::*_SEND_CYCFRAME_BY_TASKID* **SendCycFramesByTaskId**

struct *EC_T_USER_JOB_PARMS*::*_PROCESS_RXFRAME_BY_TASKID* **ProcessRxFramesByTaskId**

struct *EC_T_USER_JOB_PARMS*::*_SWITCH_EOE_FRAMES* **SwitchEoeFrames**

struct *EC_T_USER_JOB_PARMS*::*_START_TASK* **StartTask**

struct *EC_T_USER_JOB_PARMS*::*_STOP_TASK* **StopTask**

struct **_PROCESS_RXFRAME_BY_TASKID**

Public Members

EC_T_BOOL **bCycFramesProcessed**

EC_T_DWORD **dwTaskId**

struct **_SEND_CYCFRAME_BY_TASKID**

Public Members

EC_T_DWORD **dwTaskId**

struct **_START_TASK**

Public Members

EC_T_DWORD **dwTaskId**

struct **_STOP_TASK**

Public Members

EC_T_DWORD **dwTaskId**

struct **_SWITCH_EOE_FRAMES**

Public Members

EC_T_DWORD **dwMaxPortsToProcess**

EC_T_DWORD **dwNumFramesProcessed**

Detailed job description:

1. ***eUsrJob_ProcessAllRxFrames***
   If the Real-time Ethernet Driver operates in polling mode this call will process all currently received frames, in interrupt mode all received frames are processed immediately and this call just returns with nothing done.

   ```
   pUserJobParms->bAllCycFramesProcessed
   ```

   If this flag is set to a value of `EC_TRUE` it indicates that all previously initiated cyclic frames (*eUsrJob_SendAllCycFrames*) are received and processed within this call. Not used if pUserJobParms set to `EC_NULL`.

   Return: *EC_E_NOERROR* if successful, error code in case of failures.

2. ***eUsrJob_SimulatorTimer***
   This has to be called cyclically to trigger the SubDevice state machines, copy process data from simulated SubDevice firmware as well as the mailbox handling. The SubDevice state machines handle the EtherCAT® state transitions.

   Return: *EC_E_NOERROR* if successful, error code in case of failures.

6.4.16 esGetMasterState

EC_T_STATE **esGetMasterState** (EC_T_DWORD dwInstanceID)
Get the EtherCAT master current state.

Parameters
dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns
EtherCAT master state

6.4.17 esIoCtl

EC_T_DWORD **esIoCtl** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwCode,
const EC_T_VOID *const pbyInBuf,
EC_T_DWORD dwInBufSize,
EC_T_VOID *const pbyOutBuf,
EC_T_DWORD dwOutBufSize,
EC_T_DWORD *const pdwNumOutData
)
A generic control interface between the application, the EtherCAT stack and its Link Layers.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwCode** – [in] IOCTL code (EC_IOCTL…)
- **pbyInBuf** – [in] IOCTL input parameters
- **dwInBufSize** – [in] Size of IOCTL input parameters in bytes
- **pbyOutBuf** – [out] Buffer for IOCTL output
- **dwOutBufSize** – [in] Size of buffer at pbyOutBuf in bytes
- **pdwNumOutData** – [out] Amount of bytes written to pbyOutBuf by IOCTL. EC_NULL: amount not set by IOCTL.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer
- *EC_E_NOMEMORY* if memory cannot be allocated
- *EC_E_ADS_IS_RUNNING* if the ADS server is running

6.4.18 esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE

EC_IOCTL_GET_PDMEMORYSIZE
Get the process data image size. This information may be used to provide process data image storage from outside the core. This IOCTL is to be called after network configuration.

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to memory where the memory size information will be stored (type: *EC_T_MEMREQ_DESC*)
- **dwOutBufSize** – [in] Size of the output buffer in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns
EC_E_NOERROR or error code

struct **EC_T_MEMREQ_DESC**

Public Members

EC_T_DWORD **dwPDOutSize**
Size of the output process data image

EC_T_DWORD **dwPDInSize**
Size of the input process data image

See also:

esIoCtl()

6.4.19 esIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB

EC_IOCTL_REGISTER_CYCFRAME_RX_CB
Register callback for all cyclic frames received. Typically this is used when the Real-time Ethernet Driver operates in interrupt mode to get an event when the new input data (cyclic frame) is available. The callback function has to be registered after the stack initialization and before starting the job task.

Parameters

- **pbyInBuf** – [in] Cyclic frame received callback descriptor (*EC_T_CYCFRAME_RX_CBDESC*)
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns
EC_E_NOERROR or error code

struct **EC_T_CYCFRAME_RX_CBDESC**

Public Members

EC_T_VOID ***pCallbackContext***
[in] Context pointer. This pointer is used as parameter every time the callback function is called.

EC_PF_CYCFRAME_RECV ***pfnCallback***
[in] This function will be called after the cyclic frame is received, if there is more than one cyclic frame after the last frame. The application has to assure that these functions will not block.

typedef EC_T_VOID (***EC_PF_CYCFRAME_RECV***)(EC_T_DWORD dwTaskId, EC_T_VOID *pvContext)

Param dwTaskId
[in] Task id of the received cyclic frame

Param pvContext
[in] Context pointer. This pointer is used as parameter every time when the callback function is called.

See also:

esIoCtl()

6.4.20 esIoCtl - EC_IOCTL_ISLINK_CONNECTED

EC_IOCTL_ISLINK_CONNECTED
Determine whether the main link or redundancy link is connected.

Note: See also EC_IOCTL_IS_MAIN_LINK_CONNECTED, EC_IOCTL_IS_RED_LINK_CONNECTED

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to EC_T_DWORD or EC_T_LINK_CONNECTED_INFO. EC_T_DWORD: If value is EC_TRUE link is connected, if EC_FALSE it is not.
- **dwOutBufSize** – [in] Size of the output buffer in bytes (sizeof(EC_T_DWORD) / sizeof(EC_T_LINK_CONNECTED_INFO))
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns
EC_E_NOERROR or error code

Important: With cable redundancy support enabled, EC_FALSE is only set if all links are down.

See also:

esIoCtl()

6.4.21 esIoCtl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO

EC_IOCTL_GET_CYCLIC_CONFIG_INFO
Get cyclic configuration information from ENI file.

Parameters

- **pbyInBuf** – [in] Pointer to dwCycEntryIndex: Cyclic entry index for which to get information
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Pointer to *EC_T_CYC_CONFIG_DESC* data type
- **dwOutBufSize** – [in] Size of the output buffer provided at pbyOutBuf in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns
EC_E_NOERROR or error code

struct **EC_T_CYC_CONFIG_DESC**

Public Members

EC_T_DWORD **dwNumCycEntries**
[out] Total number of cyclic entries

EC_T_DWORD **dwTaskId**
[out] Task ID of selected cyclic entry (ENI: Cyclic/TaskId)

EC_T_DWORD **dwPriority**
[out] Priority of selected cyclic entry

EC_T_DWORD **dwCycleTime**
[out] Cycle time of selected cyclic entry

See also:

esIoCtl()

6.4.22 esIoCtl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES
Enable or disable additional variables for specific data types. Default: Enabled.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_BOOL. EC_TRUE: enable, EC_FALSE: disable.
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns
EC_E_NOERROR or error code

See also:

esIoCtl()

6.4.23 esIoCtl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL

EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL
Enable or disable mailbox processing at slave. Default: Enabled.

Parameters

- **pbyInBuf** – [in] Pointer to value of *EC_T_SIMULATOR_MBX_PROCESS_CTL_DESC*
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns
EC_E_NOERROR or error code

struct **EC_T_SIMULATOR_MBX_PROCESS_CTL_DESC**

Public Members

EC_T_WORD **wCfgFixedAddress**
Slave's station address. 0: all slaves.

EC_T_BOOL **bReadMbxOutEnabled**
Read mailbox out sync manager from DPRAM (received mailbox data from the master)

EC_T_BOOL **bProcessMbxEnabled**
Process mailbox data from the master (MailboxServiceInd)

See also:

esIoCtl()

6.4.24 esIoCtl - EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL

EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL
Get the information if mailbox processing at slave is enabled or disabled.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_WORD. Configured station address of slave.
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Pointer to value of *EC_T_SIMULATOR_MBX_PROCESS_CTL_DESC*
- **dwOutBufSize** – [in] Size of the output buffer provided at pbyOutBuf in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns
EC_E_NOERROR or error code

See also:

esIoCtl()

6.4.25 esIoCtl - EC_IOCTL_GET_LINKLAYER_MODE

EC_IOCTL_GET_LINKLAYER_MODE
Get Real-time Ethernet Driver mode (EcLinkMode_POLLING, EcLinkMode_INTERRUPT)

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to struct *EC_T_LINKLAYER_MODE_DESC*
- **dwOutBufSize** – [in] Size of the output buffer in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns
EC_E_NOERROR or error code

struct **EC_T_LINKLAYER_MODE_DESC**

Public Members

EC_T_LINKMODE **eLinkMode**
[out] Operation mode of main interface

EC_T_LINKMODE **eLinkModeRed**
[out] Operation mode of redundancy interface

See also:

esIoCtl()

6.4.26 esIoCtl - EC_LINKIOCTL_XXXX

The generic control interface *esIoCtl()* provides access to the main network adapter when adding EC_IOCTL_LINKLAYER_MAIN to the EC_LINKIOCTL_XXXX parameter at dwCode. EC_IOCTL_LINKLAYER_RED specifies the redundant network adapter.

Parameters pbyInBuf, dwInBufSize, pbyOutBuf, dwOutBufSize, pdwNumOutData are specific to EC_LINKIOCTL_XXXX, e.g. EC_LINKIOCTL_GET_ETHERNET_ADDRESS.

6.4.27 esIoCtl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Get MAC address of the network adapter.

esIoCtl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to MAC address buffer (6 bytes)
- `dwOutBufSize`: [in] Size of the output buffer provided at pbyOutBuf in bytes
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD: amount of bytes written to the output buffer

See also:

esIoCtl()

6.4.28 esIoCtl - EC_LINKIOCTL_GET_SPEED

Get network adapter's speed in MBits.

esIoCtl - EC_LINKIOCTL_GET_SPEED

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to EC_T_DWORD. Set by Real-time Ethernet Driver to 10/100/1000.
- `dwOutBufSize`: [in] Size of the output buffer provided at pbyOutBuf in bytes
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD: amount of bytes written to the output buffer

See also:

esIoCtl()

6.4.29 esIoCtl - EC_LINKIOCTL_GET_PCI_INFO

Get network adapter's PCI information

esIoCtl - EC_LINKIOCTL_GET_PCI_INFO

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to EC_T_PCI_INFO buffer
- `dwOutBufSize`: [in] Size of the output buffer in bytes. Must be at least the size of EC_T_PCI_INFO.
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

struct **EC_T_PCI_INFO**

Public Members

EC_T_PCI_INFO_LOCATION **Location**
PCI location (bus, device, function)

EC_T_PCI_INFO_IDENIFICATION **Ident**
PCI identification (vendor id, device id)

EC_T_DWORD **dwIoBarCnt**
I/O bar count

EC_T_PCI_INFO_IOBAR **aIoBar**
PCI I/O bars info

EC_T_DWORD **dwMemBarCnt**
Memory bar count

EC_T_PCI_INFO_MEMBAR **aMemBar**
PCI Memory bars info

EC_T_DWORD **dwInterruptCnt**
IRQ count

EC_T_PCI_INFO_INTERRUPT **aInterrupt**
PCI IRQ info

EC_T_DWORD **adwReserved**[16]
Reserved for future use

struct **EC_T_PCI_INFO_LOCATION**

Public Members

EC_T_DWORD **dwDomainNumber**
Domain

EC_T_DWORD **dwBusNumber**
Bus number

EC_T_DWORD **dwDeviceNumber**
Device number

EC_T_DWORD **dwFunctionNumber**
Function number

struct **EC_T_PCI_INFO_IDENIFICATION**

Public Members

EC_T_DWORD **dwVendorId**
Vendor ID

EC_T_DWORD **dwDeviceId**
Device ID

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_IOBAR**

Public Members

EC_T_UINT64 **qwBaseAddress**
I/O bar base address

EC_T_DWORD **dwLen**
Bar length

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_MEMBAR**

Public Members

EC_T_UINT64 **qwBaseAddress**
Memory bar base address

EC_T_DWORD **dwLen**
Bar length

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_INTERRUPT**

Public Members

EC_T_DWORD **dwIrq**
IRQ number

EC_T_CPUSET **CpuAffinity**
Reserved for future use

EC_T_DWORD **adwReserved**[2]
Reserved for future use

See also:

esIoCtl()

6.4.30 esIoCtl - EC_LINKIOCTL_FORCE_LINK_STATUS

Forces the network adapter's link status

esIoCtl - EC_LINKIOCTL_FORCE_LINK_STATUS

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_BYTE. See EC_LINK_FORCE_LINK_STATUS_…
- `dwInBufSize`: [in] Size of the input buffer in bytes. Must be at least the size of EC_T_BYTE.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

EC_LINK_FORCE_LINK_STATUS_DISCONNECTED
Force link status Disconnected: 0

EC_LINK_FORCE_LINK_STATUS_OK
Force link status OK: 1

EC_LINK_FORCE_LINK_STATUS_AUTO
Force link status Auto: 2

See also:

esIoCtl()

6.4.31 esGetVersion

EC_T_DWORD **esGetVersion** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD *pdwVersion,
EC_T_DWORD *pdwVersionType
)

Gets the version information.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwVersion** – [out] Pointer to EC_T_DWORD to carry out version number as a 32-bit value
- **pdwVersionType** – [out] Pointer to EC_T_DWORD to carry out version type. See EC_VERSION_TYPE.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL

6.4.32 esGetText

const EC_T_CHAR *__esGetText__ (EC_T_DWORD dwInstanceID, EC_T_DWORD dwTextId)

Return text tokens by ID.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

Textual description of the given ID

6.4.33 esGetMemoryUsage

EC_T_DWORD **esGetMemoryUsage** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD *pdwCurrentUsage,
EC_T_DWORD *pdwMaxUsage
)

Returns information about memory usage.

All calls to malloc/free and new/delete are monitored.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwCurrentUsage** – [out] Current memory usage in Bytes at the time where this function is called

- **pdwMaxUsage** – [out] Maximum memory usage in Bytes since initialization at the time where this function is called

Returns

EC_E_NOERROR or error code

6.4.34 esLogFrameEnable

EC_T_DWORD **esLogFrameEnable** (
EC_T_DWORD dwInstanceID,
EC_T_PFLOGFRAME_CB pvLogFrameCallBack,
EC_T_VOID *pvContext
)

Setup a callback function to log the EtherCAT network traffic.

The callback function is called by the cyclic task. Therefore the code inside the callback has to be fast and non-blocking. The callback parameter dwLogFlags can be used as a filter to log just specific frames. The master discards the frame if the callback function modifies the Ethernet frame type at byte offset 12.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvLogFrameCallBack** – [in] Pointer to frame logging callback function
- **pvContext** – [in] Pointer to function specific context

Returns

EC_E_NOERROR or error code

```
/*****************************************************************************/
/** \brief  Handler to log frames.
*
*   CAUTION: Called by cyclic task! Do not consume too much CPU time!
*/
EC_T_VOID LogFrameHandler(EC_T_VOID* pvContext, EC_T_DWORD dwLogFlags, EC_T_DWORD␣
↪dwFrameSize, EC_T_BYTE* pbyFrame)
{
    EC_T_STATE          eMasterState;

    /* get MainDevice state */
    eMasterState = (EC_T_STATE)(dwLogFlags & EC_LOG_FRAME_FLAG_MASTERSTATE_MASK);

    /* skip tx frame */
    if ((S_dwLogFrameLevel == 3) && !(dwLogFlags & EC_LOG_FRAME_FLAG_RX_FRAME))
        return;

    /* skip cyclic frame */
    if ((S_dwLogFrameLevel == 2) && !(dwLogFlags & EC_LOG_FRAME_FLAG_ACYC_FRAME))
        return;

    /* skip red frame */
    if (dwLogFlags & EC_LOG_FRAME_FLAG_RED_FRAME)
return;

    /* do something with pbyFrame ... */
}
```

6.4.35 esLogFrameDisable

EC_T_DWORD **esLogFrameDisable** (EC_T_DWORD dwInstanceID)
Disable the frame logging callback.

Parameters
dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns
EC_E_NOERROR or error code

6.5 Process Data Access Functions

6.5.1 esGetProcessImageInputPtr

EC_T_BYTE ***esGetProcessImageInputPtr** (EC_T_DWORD dwInstanceID)
Gets the process data input image pointer.

Parameters
dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns
Process data input image pointer

The size of the process data image input area is returned by *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE* and *esRegisterClient()* at *EC_T_REGISTERRESULTS::dwPDInSize*.

See also:

- *esConfigureNetwork()*
- *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- *esIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB*
- *esExecJob()* (eUsrJob_ProcessAllRxFrames) in case of polling mode

6.5.2 esGetProcessImageOutputPtr

EC_T_BYTE ***esGetProcessImageOutputPtr** (EC_T_DWORD dwInstanceID)
Gets the process data output image pointer.

Parameters
dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns
Process data output image pointer

The size of the process data image output area is returned by *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE* and *esRegisterClient()* at *EC_T_REGISTERRESULTS::dwPDOutSize*.

See also:

- *esConfigureNetwork()*
- *esIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- *esIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB*
- *esExecJob()* (eUsrJob_ProcessAllRxFrames) in case of polling mode

6.5.3 EC_COPYBITS

EC_COPYBITS (pbyDst, nDstBitOffs, pbySrc, nSrcBitOffs, nBitSize)
Copies a block of bits from a source buffer to a destination buffer.

Note: The memory buffers must be allocated before. The buffers must be big enough to hold the block starting at the given offsets! The buffers are not checked for overrun.

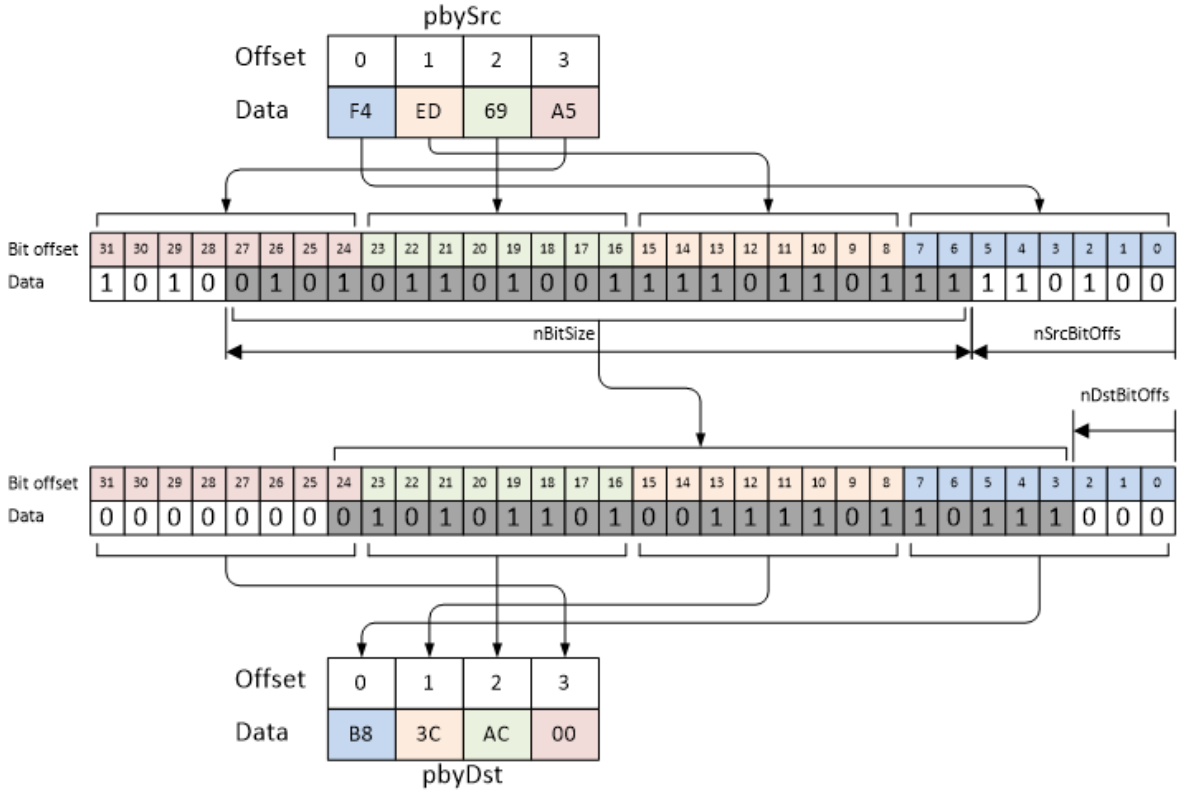
Parameters

- **pbyDst** – [out] Destination buffer
- **nDstBitOffs** – [in] Bit offset within destination buffer
- **pbySrc** – [in] Source buffer
- **nSrcBitOffs** – [in] Bit offset within source buffer
- **nBitSize** – [in] Block size in bits

See also:

- *EC_SETBITS*
- *EC_GETBITS*

-




```
EC_T_BYTE pbySrc[] = {0xF4, 0xED, 0x69, 0xA5};
EC_T_BYTE pbyDst[] = {0x00, 0x00, 0x00, 0x00};
EC_COPYBITS(pbyDst, 3, pbySrc, 6, 22);

/* pbyDst now contains 0xB8 0x3C 0xAC 0x00 */
```

6.5.4 EC_GET_FRM_WORD

EC_GET_FRM_WORD (ptr)
Reads a value of type EC_T_WORD (16 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Source buffer

Returns
EC_T_WORD value (16 bit) from buffer

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_WORD wResult = 0;

wResult = EC_GET_FRM_WORD(byFrame);
/* wResult is 0xF401 on little endian systems */

wResult = EC_GET_FRM_WORD(byFrame + 5);
/* wResult is 0x6000 on little endian systems */

wResult = EC_GET_FRM_WORD(byFrame + 2);
/* wResult is 0x85DD on little endian systems */
```

6.5.5 EC_GET_FRM_DWORD

EC_GET_FRM_DWORD (ptr)
Reads a value of type EC_T_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Source buffer

Returns
EC_T_DWORD value (32 bit) from buffer

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_DWORD dwResult = 0;

dwResult = EC_GET_FRM_DWORD(byFrame);
/* dwResult is 0x85DDF401 on little endian systems */

dwResult = EC_GET_FRM_DWORD(byFrame + 5);
/* dwResult is 0x00C16000 on little endian systems */

dwResult = EC_GET_FRM_DWORD(byFrame + 2);
/* dwResult is 0x000385DD on little endian systems */
```

6.5.6 EC_GET_FRM_QWORD

EC_GET_FRM_QWORD (ptr)
Reads a value of type EC_T_QWORD (64 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Source buffer

Returns
EC_T_QWORD value (64 bit) from buffer

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_UINT64 ui64Result = 0;

ui64Result = EC_GET_FRM_QWORD(byFrame + 1);
/* wResult is 0x00C160000385DDF4 on little endian systems */
```

6.5.7 EC_SET_FRM_WORD

EC_SET_FRM_WORD (ptr, w)
Writes a value of type EC_T_WORD (16 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Destination buffer
- **w** – [in] 16 bit value

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_WORD(byFrame + 1, 0x1234);
/* byFrame = FF 34 12 FF FF FF ... */
```

6.5.8 EC_SET_FRM_DWORD

EC_SET_FRM_DWORD (ptr, dw)
Writes a value of type EC_T_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Destination buffer
- **dw** – [in] 32 bit value

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_DWORD(byFrame + 1, 0x12345678);
/* byFrame = FF 78 56 34 12 FF ... */
```

6.5.9 EC_SET_FRM_QWORD

EC_SET_FRM_QWORD (ptr, qw)
Writes a value of type EC_T_QWORD (64 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Destination buffer
- **qw** – [in] 64 bit value

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_QWORD(byFrame + 1, 0xFEDCBA9876543210);
/* byFrame = FF 10 32 54 76 98 BA DC FE FF FF ... */
```

6.5.10 EC_GETBITS

EC_GETBITS (pbySrcBuf, pbyDstData, nSrcBitOffs, nBitSize)
Reads a given number of bits from source buffer starting at given bit offset to destination buffer.

Note: This function should only be used to get bit-aligned data. For byte-aligned data the corresponding functions should be used.

Parameters

- **pbySrcBuf** – [in] Source buffer to be copied
- **pbyDstData** – [out] Destination buffer where data is copied to
- **nSrcBitOffs** – [in] Source bit offset where data is copied from
- **nBitSize** – [in] Bit count to be copied

See also:

- *EC_GET_FRM_WORD*
- *EC_GET_FRM_DWORD*
- *EC_GET_FRM_QWORD*

6.5.11 EC_SETBITS

EC_SETBITS (pbyDstBuf, pbySrcData, nDstBitOffs, nBitSize)
Writes a given number of bits from source data starting at first bit to destination buffer at given bit offset.

Note: This function should only be used to set bit-aligned data. For byte-aligned data the corresponding functions should be used.

Parameters

- **pbyDstBuf** – [out] Destination buffer where data is copied to

- **pbySrcData** – [in] Source buffer to be copied, starting with first bit
- **nDstBitOffs** – [in] Destination bit offset where data is copied to
- **nBitSize** – [in] Bit count to be copied

See also:

- *EC_SET_FRM_WORD*
- *EC_SET_FRM_DWORD*
- *EC_SET_FRM_QWORD*

6.5.12 EC_COPYBIT

EC_COPYBIT(pbyBuf, nBitOffs, bVal)
Copy a boolean bit value into the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to write.
- **bVal** – Boolean bit value.

6.5.13 EC_TESTBIT

EC_TESTBIT(pbyBuf, nBitOffs)
Test whether a bit in the buffer is set.

Parameters

- **pbyBuf** – Source buffer.
- **nBitOffs** – Bit offset to test.

Returns
EC_TRUE if bit is set, otherwise EC_FALSE.

6.5.14 EC_SETBIT

EC_SETBIT(pbyBuf, nBitOffs)
Set a single bit in the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to set.

6.5.15 EC_CLRBIT

EC_CLRBIT (pbyBuf, nBitOffs)
Clear a single bit in the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to clear.

6.5.16 esGetProcessData

EC_T_DWORD **esGetProcessData** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bOutputData,
EC_T_DWORD dwOffset,
EC_T_BYTE *pbyData,
EC_T_DWORD dwDataLen,
EC_T_DWORD dwTimeout
)
Blocking function to retrieve consistent process data from outside the JobTask context.

This function requests a copy of the process data (stored in RAM). The actual memcpy operation is executed by the JobTask to ensure data consistency. While waiting for the copy to complete, the calling context blocks and repeatedly calls sleep with an interval of at least the cycle time or one millisecond, whichever is greater. The function returns either with the requested process data once the copy is finished, or when the specified timeout has expired.

Note: The function is blocking and should be used carefully in time-sensitive contexts and may not be called from within the JobTask context. If process data are required outside the cyclic simulator job task (which is calling esExecJob), direct access to the process data is not recommended as data consistency cannot be guaranteed. A call to this function will send a data read request to the simulator stack and then check every millisecond whether new data is provided. The simulator stack will provide new data after calling esExecJob(eUsrJob_SimulatorTimer) within the job task. This function is usually only called remotely (using the Remote API).

Note: This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: read output data, EC_FALSE: read input data
- **dwOffset** – [in] Byte offset in Process data to read from
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **dwTimeout** – [in] Timeout [ms]

Returns
EC_E_NOERROR or error code

6.5.17 esSetProcessData

EC_T_DWORD **esSetProcessData** (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwOffset,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)

Write Process data synchronized.

If process data shall be set outside the cyclic simulator job task (which is calling esExecJob), direct access to the process data is not recommended as data consistency cannot be guaranteed. A call to this function will send a data write request to the simulator stack and then check every millisecond whether new data is written. The simulator stack will copy the data after calling esExecJob(eUsrJob_SimulatorTimer) within the job task. This function is usually only called remotely (using the Remote API).

Note: This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: write output data, EC_FALSE: write input data
- **dwOffset** – [in] Byte offset in Process data to write to
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **dwTimeout** – [in] Timeout [ms]

Returns

EC_E_NOERROR or error code

6.5.18 esSetProcessDataBits

EC_T_DWORD **esSetProcessDataBits** (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataBitLen,
    EC_T_DWORD dwTimeout
)

Writes a specific number of bits from a given buffer to the process image with a bit offset (synchronized).

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: write output data, EC_FALSE: write input data
- **dwBitOffsetPd** – [in] Bit offset in Process data image
- **pbyData** – [in] Buffer containing transferred data

- **dwDataBitLen** – [in] Buffer length [bit]
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC_NOWAIT.

Returns
EC_E_NOERROR or error code

6.5.19 esGetProcessDataBits

EC_T_DWORD **esGetProcessDataBits** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bOutputData,
EC_T_DWORD dwBitOffsetPd,
EC_T_BYTE *pbyData,
EC_T_DWORD dwDataBitLen,
EC_T_DWORD dwTimeout
)

Reads a specific number of bits from the process image to the given buffer with a bit offset (synchronized).

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: read output data, EC_FALSE: write input data
- **dwBitOffsetPd** – [in] Bit offset in Process data image
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataBitLen** – [in] Buffer length [bit]
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC_NOWAIT.

Returns
EC_E_NOERROR or error code

6.6 Notifications

6.6.1 Notification handler

The application can register a notification handler using `esRegisterClient()` for notifications as described below.

A further important rule exists due to the fact that this callback function is usually called in the context of the EC-Simulator stack timer thread. As the whole EtherCAT® operation is blocked while calling this function the notification handler must not use much CPU time or even call operating system functions that may block. Time consuming operations should be executed in separate application threads.

Data structure filled with detailed information about the according notification:

struct **EC_T_NOTIFYPARMS**
Data structure filled with detailed information about the according notification.

Public Members

EC_T_VOID *pCallerData
[in] Client depending caller data parameter. This pointer is one of the parameters when the client registers.

EC_T_BYTE *pbyInBuf
[in] Notification input parameters

EC_T_DWORD dwInBufSize
[in] Size of notification input parameters in bytes

EC_T_BYTE *pbyOutBuf
[out] Buffer for notification output (result)

EC_T_DWORD dwOutBufSize
[in] Size of buffer at pbyOutBuf in bytes

EC_T_DWORD *pdwNumOutData
[out] Amount of bytes written to pbyOutBuf by notification. EC_NULL: amount not set by notification.

6.6.2 EC_NOTIFY_SLAVE_PRESENCE

This notification is given when a SubDevice appears or disappears from the network.

emNotify - EC_NOTIFY_SLAVE_PRESENCE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVE_PRESENCE_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Disconnecting the SubDevice from the network, powering it off or a bad connection will generate this notification.

struct **EC_T_SLAVE_PRESENCE_NTFY_DESC**

Public Members

EC_T_WORD wStationAddress
Slave station address

EC_T_BYTE bPresent
EC_TRUE: present, EC_FALSE: absent

6.6.3 EC_NOTIFY_SLAVE_STATECHANGED

This notification is given when a SubDevice changed its EtherCAT® state.

emNotify - EC_NOTIFY_SLAVES_STATECHANGED

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVE_STATECHANGED_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_SLAVE_STATECHANGED_NTFY_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_STATE **newState**
New slave state

6.6.4 EC_NOTIFY_STATECHANGED

This notification is given when the MainDevice changes its EtherCAT® state.

emNotify - EC_NOTIFY_STATECHANGED

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_STATECHANGE which contains the old and the new MainDevice operational state
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_STATECHANGE**

Public Members

EC_T_STATE **oldState**
Old operational state

EC_T_STATE **newState**
New operational state

6.6.5 EC_NOTIFY_HC_TOPOCHGDONE

This notification is given by eUsrJob_SimulatorTimer when the topology changes.

emNotify - EC_NOTIFY_HC_TOPOCHGDONE

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

See also:

esPowerSlave() , *esDisconnectPort()* , *esConnectPorts()*

6.6.6 EC_NOTIFY_SLAVE_ERROR_STATUS_INFO

This notification is given when the Error bit on the specific SubDevice is set or cleared Detailed error information is stored in structure *EC_T_SLAVE_ERROR_INFO_DESC* of `EC_T_ERROR_NOTIFICATION_DESC`.

struct **EC_T_SLAVE_ERROR_INFO_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_WORD **wStatus**
Slave Status (AL Status)

EC_T_WORD **wStatusCode**
Error status code (AL STATUS CODE)

6.6.7 EC_NOTIFY_EEPROM_OPERATION

This notification is given, when a SubDevice EEPROM is written by the MainDevice.

emNotify - EC_NOTIFY_EEPROM_OPERATION

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_EEPROM_OPERATION_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_EEPROM_OPERATION_NTFY_DESC**

Public Members

EC_T_DWORD **dwTferId**
Transfer ID. For every new EEPROM operation a unique ID has to be assigned. This ID can be used after completion to identify the transfer.

EC_T_EEPROM_OPERATION_TYPE **eType**
Type of EEPROM operation

EC_T_DWORD **dwResult**
Result of EEPROM operation

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

union **_EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT**

struct **_EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT_ACTIVE**

Public Members

EC_T_BOOL **bSlavePDIAccessActive**
EC_TRUE: EEPROM active by PDI application, EC_FALSE: EEPROM not active

struct **_EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT_READ**

Public Members

EC_T_WORD **wEEPRomStartOffset**
Start address of EEPRom operation. Given by API.

EC_T_WORD ***pwData**
Pointer to WORD array containing the data. Given by API.

EC_T_DWORD **dwReadLen**
Number of Words to be read. Given by API.

EC_T_DWORD **dwNumOutData**
Number of Words actually read from EEPRom

struct **_EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT_WRITE**

Public Members

EC_T_WORD **wEEPRomStartOffset**
Start address of EEPRom operation. Given by API.

EC_T_WORD ***pwData**
Pointer to WORD array containing the data. Given by API.

EC_T_DWORD **dwWriteLen**
Number of Words to be written. Given by API.

enum **EC_T_EEPROM_OPERATION_TYPE**
Values:

enumerator **eEEPRomOp_Unknown**
Unknown EEPROM operation, only for internal use

enumerator **eEEPRomOp_Assign**
Assign slave EEPROM operation, used by emAssignSlaveEEPRomReq

enumerator **eEEPRomOp_Active**
Active slave EEPROM operation, used by emActiveSlaveEEPRomReq

enumerator **eEEPRomOp_Read**
Read slave EEPRom operation, used by emReadSlaveEEPRomReq

enumerator **eEEPRomOp_Write**
Write slave EEPRom operation, used by emWriteSlaveEEPRomReq

enumerator **eEEPRomOp_Reload**
Reload slave EEPRom operation, used by emReloadSlaveEEPRomReq

enumerator **eEEPRomOp_Reset**
Reset slave EEPRom operation, used by emResetSlaveController

6.6.8 EC_NOTIFY_FRAME_RESPONSE_ERROR

This notification is given, when a Ethernet pause frame (MAC control EtherType 0x8808, Control opcode 0x0001) is received. This may occur if any Flow Control mode is active within the network adapter's native driver settings.

emNotify - EC_NOTIFY_FRAME_RESPONSE_ERROR

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_ERROR_NOTIFICATION_DESC
- `dwInBufSize`: [in] SIZEOF_EC_T_ERROR_NOTIFICATION_FRAME_RSPERR
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_FRAME_RSPERR_DESC**

Public Members

EC_T_BOOL **bIsCyclicFrame**
Indicates whether the lost frame was a cyclic frame

EC_T_FRAME_RSPERR_TYPE **EErrorType**
Frame response error type

EC_T_BYTE **byEcCmdHeaderIdxSet**
Expected IDX value, this value is valid only for acyclic frames in case EErrorType is not equal to eRspErr_UNEXPECTED

EC_T_BYTE **byEcCmdHeaderIdxAct**
Actually received IDX value, this value is only valid for acyclic frames in case of EErrorType is equal to: eRspErr_WRONG_IDX and eRspErr_UNEXPECTED

EC_T_WORD **wCycFrameNum**
Number of the lost cyclic frame from the ENI

EC_T_DWORD **dwTaskId**
Cyclic Task ID (ENI: Cyclic/TaskId). Only valid if bIsCyclicFrame is set

enum **EC_T_FRAME_RSPERR_TYPE**
Values:

enumerator **eRspErr_UNDEFINED**
Undefined

enumerator **eRspErr_NO_RESPONSE**
No Ethernet frame received (timeout, frame loss)

enumerator **eRspErr_WRONG_IDX**
Wrong IDX value in acyclic frame

enumerator **eRspErr_UNEXPECTED**
Unexpected frame was received

enumerator **eRspErr_FRAME_RETRY**
Ethernet frame will be re-sent (timeout, frame loss)

enumerator **eRspErr_RETRY_FAIL**
All retry mechanism fails to re-sent acyclic frames

enumerator **eRspErr_FOREIGN_SRC_MAC**
Frame with MAC from other Master received

enumerator **eRspErr_NON_ECAT_FRAME**
Non EtherCAT frame received

enumerator **eRspErr_CRC**
Ethernet frame with CRC error received

enumerator **eRspErr_ETHERNET_PAUSE_FRAME**
Ethernet pause frame (flow control) received

6.6.9 esNotifyApp

EC_T_DWORD **esNotifyApp** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwCode,
EC_T_NOTIFYPARMS *pParms
)
Calls the notification callback functions of all registered clients.

Note: EC_E_ERROR and EC_E_INVALIDPARM from registered clients' callback functions are ignored.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwCode** – [in] Application specific notification code. dwCode must be <= EC_NOTIFY_APP_MAX_CODE. The callback functions get "EC_NOTIFY_APP | dwCode" as parameter.
- **pParms** – [in] Parameter to all callback functions. Note: Output parameters are not transferred from RAS client to RAS server.

Returns
EC_E_ERROR or first error code different from *EC_E_ERROR* and *EC_E_INVALIDPARM* of registered clients' callback functions

6.7 Network operation functions

6.7.1 esConnectPorts

EC_T_DWORD **esConnectPorts** (
    EC_T_DWORD dwInstanceId1,
    EC_T_WORD wCfgFixedAddress1,
    EC_T_BYTE byPort1,
    EC_T_DWORD dwInstanceId2,
    EC_T_WORD wCfgFixedAddress2,
    EC_T_BYTE byPort2
)

Connect a slave ESC port to another slave ESC port or to a network adapter.

–

Parameters

- **dwInstanceId1** – [in] Simulator Instance ID of Port1
- **wCfgFixedAddress1** – [in] Slave's station address of Port1. 0: network adapter
- **byPort1** – [in] ESC port of Port1. 0, 1, 2, 3: port A, port B, port C, port D.
- **dwInstanceId2** – [in] Simulator Instance ID of Port2
- **wCfgFixedAddress2** – [in] Slave's station address of Port2. 0: network adapter
- **byPort2** – [in] ESC port of Port2. 0, 1, 2, 3: port A, port B, port C, port D.

Returns

EC_E_NOERROR or error code

6.7.2 esDisconnectPort

EC_T_DWORD **esDisconnectPort** (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    EC_T_BYTE byPort
)

Disconnect a slave ESC port or a network adapter.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: network adapter.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.

Returns

EC_E_NOERROR or error code

6.7.3 esPowerSlave

EC_T_DWORD **esPowerSlave** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BOOL bOn
)

Power on or power off a slave.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **bOn** – [in] EC_FALSE: switch off, EC_TRUE: switch on

Returns

EC_E_NOERROR or error code

6.8 Error simulation functions

6.8.1 esSetErrorAtSlavePort

EC_T_DWORD **esSetErrorAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort,
EC_T_BOOL bOutgoing
)

Trigger loss of sent or received EtherCAT frames at slave port ("single shot").

–

Remark

See ESC registers RX Error Counter (0x0300:0x0307), Forwarded RX Error Counter (0x0308:0x030B), ECAT Processing Unit Error Counter (0x030C)

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.
- **bOutgoing** – [in] Direction. EC_FALSE: Receive Frame, EC_TRUE: Send Frame.

Returns

EC_E_NOERROR or error code

6.8.2 esSetErrorGenerationAtSlavePort

EC_T_DWORD **esSetErrorGenerationAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort,
EC_T_BOOL bOutgoing,
EC_T_DWORD dwLikelihoodPpm,
EC_T_DWORD dwFixedGoodFramesCnt,
EC_T_DWORD dwFixedErroneousFramesCnt
)

Simulate the loss of sent EtherCAT frames at the port of a simulated slave at either random, periodic or random periodic intervalls.

- Random frame loss simulation: For each frame the dwLikelihoodPpm parameter determines whether the frame will be discarded.
- Periodic frame loss simulation: After dwFixedErroneousFramesCnt discarded frames, dwFixedGoodFramesCnt frames will be processed.
- Random periodic frame loss simulation: The dwLikelihoodPpm parameter determines whether a periodic frame loss sequence is triggered.

–

Remark

See ESC registers RX Error Counter (0x0300:0x0307), Forwarded RX Error Counter (0x0308:0x030B), ECAT Processing Unit Error Counter (0x030C)

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave station address. 0: All slaves
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.
- **bOutgoing** – [in] Direction. EC_FALSE: Receive Frame, EC_TRUE: Send Frame.
- **dwLikelihoodPpm** – [in] Likelihood (ppm) according to frame loss simulation mode (Random / Random Periodic). Set to 0 in case of Periodic frame loss simulation.
- **dwFixedGoodFramesCnt** – [in] Number of processed frames according to frame loss simulation mode (Periodic / Random Periodic). Set to 0 in case of Random frame loss simulation.
- **dwFixedErroneousFramesCnt** – [in] Number of discarded frames according to frame loss simulation mode (Periodic / Random Periodic). Set to 0 in case of Random frame loss simulation.

Returns

EC_E_NOERROR or error code

6.8.3 esResetErrorGenerationAtSlavePorts

EC_T_DWORD **esResetErrorGenerationAtSlavePorts** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress
)

Reset physical layer error generation destroying frames at slave port.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.

Returns

EC_E_NOERROR or error code

6.8.4 esSetLinkDownAtSlavePort

EC_T_DWORD **esSetLinkDownAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort,
EC_T_BOOL bDown,
EC_T_DWORD dwLinkDownTimeMs
)

Trigger link lost event.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.
- **bDown** – [in] Connect or disconnect cable. EC_TRUE: disconnect.
- **dwLinkDownTimeMs** – [in] Link down duration [ms] or EC_WAITINFINITE

Returns

EC_E_NOERROR or error code

6.8.5 esSetLinkDownGenerationAtSlavePort

EC_T_DWORD **esSetLinkDownGenerationAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort,
EC_T_DWORD dwLikelihoodPpm,
EC_T_DWORD dwFixedLinkDownTimeMs,
EC_T_DWORD dwFixedLinkUpTimeMs
)

Generate link lost events randomly or at fixed intervals.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.
- **dwLikelihoodPpm** – [in] Random simulation: link down likelihood (ppm) within On-Timer
- **dwFixedLinkDownTimeMs** – [in] On link down simulation: fixed link down duration [ms] (at least)
- **dwFixedLinkUpTimeMs** – [in] After link down was simulated: fixed link up duration [ms] (at least)

Returns
EC_E_NOERROR or error code

6.8.6 esResetLinkDownGenerationAtSlavePorts

EC_T_DWORD **esResetLinkDownGenerationAtSlavePorts** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress
)

Reset link lost event generation.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves

Returns
EC_E_NOERROR or error code

6.8.7 esLogFrameEnableAtSlavePort

EC_T_DWORD **esLogFrameEnableAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort,
EC_T_PFLOGFRAME_CB pvLogFrameCallBack,
EC_T_VOID *pvContext
)

Register EtherCAT network traffic logging function at given slave port.

–

Note: The simulator introduces frame errors if the callback function modifies the Ethernet frame type at byte offset 12.

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.
- **pvLogFrameCallBack** – [in] Pointer to frame logging callback function
- **pvContext** – [in] Pointer to context passed as first parameters to callback function

Returns
EC_E_NOERROR or error code

6.8.8 esLogFrameDisableAtSlavePort

EC_T_DWORD **esLogFrameDisableAtSlavePort** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_BYTE byPort
)
Unregister EtherCAT network traffic logging function from given slave port.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **byPort** – [in] ESC port. 0, 1, 2, 3: port A, port B, port C, port D.

Returns
EC_E_NOERROR or error code

6.8.9 esSendSlaveCoeEmergency

EC_T_DWORD **esSendSlaveCoeEmergency** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_WORD wCode,
EC_T_BYTE *pbyData,
EC_T_DWORD dwDataLen
)
Send CoE emergency (queued)

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **wCode** – [in] Emergency code
- **pbyData** – [in] Emergency data
- **dwDataLen** – [in] Length of emergency data in byte

Returns
EC_E_NOERROR or error code

CoE Emergency Example

The following code demonstrates how to send a CoE Emergency from a local buffer to the MainDevice.

```
    /* send CoE Emergency */
    EC_T_BYTE abyEmergencyData[6] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 };
    dwRes = esSendSlaveCoeEmergency(dwSimulatorId, wSlaveAddress, 0x1234 /* code */
↪, abyEmergencyData, 6 /* data length */);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"esSendSlaveCoeEmergency failed: %s (0x%lx))\n", esGetText(dwSimulatorId, dwRes),
↪ dwRes));
        goto Exit;
    }
```

6.8.10 esSetSimSlaveState

EC_T_DWORD **esSetSimSlaveState** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_WORD wDeviceState,
EC_T_WORD wDeviceStatusCode
)

Simulate AL Status Error for slave without device emulation or acknowledge delayed EtherCAT state change.

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **wDeviceState** – [in] Device state (DEVICE_STATE…)
- **wDeviceStatusCode** – [in] Device status code (DEVICE_STATUSCODE…)

Returns

EC_E_NOERROR or error code

Simulate AL Status Error Example

The following code demonstrates how to simulate AL Status Errors for SubDevices without network adapter emulation:

```
/* start and immediately stop AL Status Error simulation (single shot) */
dwRes = esSetSimSlaveState(dwSimulatorId, wSlaveAddr,
            DEVICE_STATE_PREOP, DEVICE_STATUSCODE_ERROR);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
dwRes = esSetSimSlaveState(dwSimulatorId, wSlaveAddr,
            0, DEVICE_STATUSCODE_NOERROR);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
```

Simulate AL Status Delay Example

The following code demonstrates how to simulate the EtherCAT® state transition being delayed for some cycles by the SubDevice application.

`APPL_StartMailboxHandlerCallback_DelayALStatus()` starts the EtherCAT® state transition delay and `APPL_ApplicationCallback_DelayALStatus()` stops the EtherCAT® state transition delay:

```
/* AL Status delay timer */
static CEcTimer S_oAlStatusDelayTimer;

/* EC_PF_SSC_APPL_START_MAILBOX_HANDLER */
static EC_T_WORD EC_FNCALL APPL_StartMailboxHandlerCallback_DelayALStatus(
    EC_T_VOID* pvContext,
    EC_T_DWORD /* dwSimulatorId */,
    EC_T_WORD /* wCfgFixedAddress */)
{
    T_EC_DEMO_APP_CONTEXT* pAppContext = (T_EC_DEMO_APP_CONTEXT*)pvContext;

    if (!S_oAlStatusDelayTimer.IsStarted())
    {
        EC_T_DWORD dwBusCycleTimeMsec = pAppContext->AppParms.dwBusCycleTimeNsec /␣
↪1000000;

        /* simulate slave state delay (e.g 3 cycles) */
        S_oAlStatusDelayTimer.Start(EC_AT_LEAST(3 * dwBusCycleTimeMsec, 1));
    }

    /* return NOERROR_INWORK to delay the EtherCAT state transition */
    return 0xFF; /* NOERROR_INWORK, see also esSetSimSlaveState(..., 0, DEVICE_
↪STATUSCODE_NOERROR) */
}

/* EC_PF_SSC_APPL_APPLICATION */
static EC_T_VOID EC_FNCALL APPL_ApplicationCallback_DelayALStatus(
    EC_T_VOID* /* pvContext */,
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress)
{
    /* application ready */
    if (S_oAlStatusDelayTimer.IsElapsed())
    {
        /* signal delayed EtherCAT state transition complete (see also NOERROR_
↪INWORK) */
        esSetSimSlaveState(dwInstanceId, wCfgFixedAddress, 0, DEVICE_STATUSCODE_
↪NOERROR);

        S_oAlStatusDelayTimer.Stop();
    }
}
```

S_oAlStatusDelayTimer simulates the delay until the SubDevice application is ready.

Note: If the delay is too long, the transition to PREOP will fail in EcSimulatorSilDemo due to InitCmd timeouts from the ENI file and checks within the SSC:

The callbacks must be registered using *esSetSlaveSscApplication()*, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo:

```
struct _EC_T_SLAVE_SSC_APPL_DESC oSlaveAppDesc;
OsMemset(&oSlaveAppDesc, 0, sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC));
```

(continues on next page)

(continued from previous page)

```
oSlaveAppDesc.dwSignature = SIMULATOR_SIGNATURE;
oSlaveAppDesc.dwSize = sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC);
oSlaveAppDesc.szName = (EC_T_CHAR*)"mySlaveAppl";
oSlaveAppDesc.pvContext = pAppContext;

/* register callback APPL_Application, called after frame processing*/
oSlaveAppDesc.pfnApplication = APPL_ApplicationCallback_DelayALStatus;
/* register callback for master state transition request INIT to higher */
oSlaveAppDesc.pfnStartMailboxHandler = APPL_StartMailboxHandlerCallback_
↪DelayALStatus;
/* register SlaveSscApplication callbacks (Master INIT) */
dwRes = esSetSlaveSscApplication(dwSimulatorId, wSlaveAddress, &oSlaveAppDesc);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
```

```
dwRes = emSetMasterState(0, ETHERCAT_STATE_CHANGE_TIMEOUT, eEcatState_PREOP);
```

See also:

- *esSetSlaveSscApplication()*: *EC_T_SLAVE_SSC_APPL_DESC::pfnAckErrorInd*

6.9 SubDevice control and status functions

6.9.1 esGetNumConfiguredSlaves

EC_T_DWORD **esGetNumConfiguredSlaves** (EC_T_DWORD dwInstanceID)

Returns the number of slaves which are configured in the ENI.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

Number of slaves

6.9.2 esGetNumConnectedSlaves

EC_T_DWORD **esGetNumConnectedSlaves** (EC_T_DWORD dwInstanceID)

Get number of currently connected slaves.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

Number of connected slaves

6.9.3 esGetSlaveId

EC_T_DWORD **esGetSlaveId** (EC_T_DWORD dwInstanceID, EC_T_WORD wStationAddress)
Determines the slave ID using the slave station address.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wStationAddress** – [in] Station address of the slave

Returns
Slave ID or INVALID_SLAVE_ID if the slave could not be found or the stack is not initialized

Example

```
/* get slave id of slave with station address 1001 */
EC_T_DWORD dwSlaveId = esGetSlaveId(dwSimulatorId, 1001);
if (INVALID_SLAVE_ID == dwSlaveId)
{
    dwRetVal = EC_E_NOTFOUND;
    goto Exit;
}
```

6.9.4 esGetSlaveIdAtPosition

EC_T_DWORD **esGetSlaveIdAtPosition** (
EC_T_DWORD dwInstanceID,
EC_T_WORD wAutoIncAddress
)
Determines the slave ID using the slave auto increment address.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wAutoIncAddress** – [in] Auto increment address of the slave

Returns
Slave ID or INVALID_SLAVE_ID if no slave matching wAutoIncAddress can be found

Example

```
/* get slave id of third slave (auto inc address 0, -1, -2, ...) */
EC_T_DWORD dwSlaveId = esGetSlaveIdAtPosition(dwSimulatorId, (EC_T_
↪WORD)-2);
if (INVALID_SLAVE_ID == dwSlaveId)
{
    dwRetVal = EC_E_NOTFOUND;
    goto Exit;
}
```

6.9.5 esGetSlaveState

EC_T_DWORD **esGetSlaveState** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwCurrDevState,
    EC_T_WORD *pwReqDevState
)

Get the slave state.

The slave state is always read automatically from the AL_STATUS register whenever necessary. It is not forced by calling this function. This function may be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pwCurrDevState** – [out] Current slave state
- **pwReqDevState** – [out] Requested slave state

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointers are EC_NULL
- *EC_E_SLAVE_NOT_PRESENT* if the slave is not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found

6.9.6 esSetSlaveState

EC_T_DWORD **esSetSlaveState** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwTimeout
)

Set a specified slave into the requested EtherCAT state.

The requested state shall not be higher than the overall operational state. DEVICE_STATE_BOOTSTRAP can only be requested if the slave's state is INIT. This function may not be called from within the JobTask's context.

If the function is called with EC_NOWAIT, the client may wait for reaching the requested state using the notification callback (EC_NOTIFY_SLAVE_STATECHANGED).

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wDeviceState** – [in] Requested device state. See Slave device state's

- **dwTimeout** – [in] Timeout [ms]. This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC_NOWAIT the function will return immediately.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized or denies the requested state, see comments below
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or BOOTSTRAP was requested for a slave that does not support it
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if the EtherCAT stack cannot execute the request at this time, the function has to be called at a later time
- *EC_E_NOTREADY* if the working counter was not set when requesting the slave's state (slave may not be connected or did not respond)
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

6.9.7 esIsSlavePresent

```
EC_T_DWORD esIsSlavePresent (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BOOL *pbPresence
)
```

Returns whether a specific slave is currently connected to the Bus.

This function may be called from within the JobTask.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pbPresence** – [out] EC_TRUE if the slave is currently connected to the bus, EC_FALSE if not

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found

6.9.8 esGetSlaveProp

EC_T_BOOL **esGetSlaveProp** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_SLAVE_PROP *pSlaveProp
)

Determines the properties of the slave device.

deprecated Use esGetCfgSlaveInfo instead

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pSlaveProp** – [out] Slave properties

Returns

EC_TRUE if the slave exists, EC_FALSE if no slave matching dwSlaveId can be found

6.9.9 esGetSlavePortState

EC_T_DWORD **esGetSlavePortState** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwPortState
)

Returns the state of the slave ports.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pwPortState** – [out] Slave port state.

  Format: wwww xxxx yyyy zzzz (each nibble : port 3210)

  wwww : Signal detected 1=yes, 0=no (ESC Register 0x110 Bit 9, 11, 13, 15)

  xxxx : Loop closed 1=yes, 0=no (ESC Register 0x110 Bit 8, 10, 12, 14)

  yyyy : Link established 1=yes, 0=no (ESC Register 0x110 Bit 4, 5, 6, 7)

  zzzz : Slave connected 1=yes, 0=no (zzzz = logical result of w,x,y)

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found

6.9.10 esGetProcessVarInfoNumOf, esGetProcessVarInfoEx

EC_T_DWORD **esGetProcessVarInfoNumOf** (
EC_T_DWORD dwInstanceID,
EC_T_VAR_DIRECTION eVarDirection,
EC_T_VAR_SOURCE eVarSource,
EC_T_BOOL bFixedAddress,
EC_T_WORD wSlaveAddress,
EC_T_DWORD *pdwProcessVarInfoNumOf
)

Get process variables information.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eVarDirection** – [in] INPUTs, OUTPUTs, … . See *EC_T_VAR_DIRECTION* .
- **eVarSource** – [in] Slave, Master, … . See *EC_T_VAR_SOURCE* .
- **bFixedAddress** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddress
- **pdwProcessVarInfoNumOf** – [out] Process variables count

Returns

EC_E_NOERROR or error code

EC_T_DWORD **esGetProcessVarInfoEx** (
EC_T_DWORD dwInstanceID,
EC_T_VAR_DIRECTION eVarDirection,
EC_T_VAR_SOURCE eVarSource,
EC_T_BOOL bFixedAddress,
EC_T_WORD wSlaveAddress,
EC_T_PROCESS_VAR_INFO_EX *aoVarInfoEx,
EC_T_DWORD dwMaxVarInfoCnt,
EC_T_DWORD *pdwVarInfoCnt
)

Get process variables information.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eVarDirection** – [in] INPUTs, OUTPUTs, … . See *EC_T_VAR_DIRECTION* .
- **eVarSource** – [in] Slave, Master, … . See *EC_T_VAR_SOURCE* .
- **bFixedAddress** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddress
- **aoVarInfoEx** – [out] The read process variable extended information entries
- **dwMaxVarInfoCnt** – [in] Maximum number of variables that can be stored at aoVarInfoEx
- **pdwVarInfoCnt** – [out] Number process variable entries that have been stored in aoVarInfoEx

Returns

EC_E_NOERROR or error code

The following example demonstrates how to get all process data variables:

esGetProcessVarInfoEx() Example

```
EC_T_DWORD dwProcessVarInfoNumOf = 0;
EC_T_DWORD dwRes = EC_E_NOERROR;

dwRes = esGetProcessVarInfoNumOf(dwSimulatorId, eVarDirection_All, eVarSource_All,␣
↪EC_FALSE, 0, &dwProcessVarInfoNumOf);
/* ... */
EC_T_PROCESS_VAR_INFO_EX* aoVarInfoEx = (EC_T_PROCESS_VAR_INFO_
↪EX*)OsMalloc(dwProcessVarInfoNumOf * sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = esGetProcessVarInfoEx(dwSimulatorId, eVarDirection_All, eVarSource_All, EC_
↪FALSE, 0, aoVarInfoEx, dwProcessVarInfoNumOf, EC_NULL);
/* ... */
OsSafeFree(aoVarInfoEx);
```

EC_T_VAR_DIRECTION

enum **EC_T_VAR_DIRECTION**
Values:

enumerator **eVarDirection_Undefined**
Undefined Direction

enumerator **eVarDirection_INPUT**
INPUTs

enumerator **eVarDirection_OUTPUT**
OUTPUTs

enumerator **eVarDirection_All**
INPUTs and OUTPUTs

enumerator **eVarDirection_BCppDummy**

EC_T_VAR_SOURCE

enum **EC_T_VAR_SOURCE**
Values:

enumerator **eVarSource_Undefined**
Undefined Source

enumerator **eVarSource_All**
Slaves and Master/Monitor/Simulator

enumerator **eVarSource_AllSlaves**
All Slaves

enumerator **eVarSource_Slave**
Slave

enumerator **eVarSource_Master**
Master

enumerator **eVarSource_Monitor**
Monitor

enumerator **eVarSource_Simulator**
Simulator

enumerator **eVarSource_BCppDummy**

6.9.11 esGetSlaveInpVarInfoNumOf

EC_T_DWORD **esGetSlaveInpVarInfoNumOf** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD *pwSlaveInpVarInfoNumOf
)
Gets the number of input variables of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pwSlaveInpVarInfoNumOf** – [out] Number of found process variable entries

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

6.9.12 esGetSlaveOutpVarInfoNumOf

EC_T_DWORD **esGetSlaveOutpVarInfoNumOf** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD *pwSlaveOutpVarInfoNumOf
)
Gets the number of output variables of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing

- **pwSlaveOutpVarInfoNumOf** – [out] Number of found process variables

Returns
EC_E_NOERROR or error code

6.9.13 esGetSlaveInpVarInfo

EC_T_DWORD **esGetSlaveInpVarInfo** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wNumOfVarsToRead,
EC_T_PROCESS_VAR_INFO *pSlaveProcVarInfoEntries,
EC_T_WORD *pwReadEntries
)

Gets the process variable information entries of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number process variable entries that have been stored in pSlaveProcVarInfoEntries
- **pSlaveProcVarInfoEntries** – [out] The read process variable information entries
- **pwReadEntries** – [out] The number of read process variable information entries

Returns
EC_E_NOERROR or error code

struct **EC_T_PROCESS_VAR_INFO**

Public Members

EC_T_CHAR **szName**[MAX_PROCESS_VAR_NAME_LEN]
[out] Name of the found process variable

EC_T_WORD **wDataType**
[out] Data type of the found process variable (according to ETG.1000, section 5). See also EcType.h, DEFTYPE_BOOLEAN.

EC_T_WORD **wFixedAddr**
[out] Station address of the slave that is owner of this variable

EC_T_INT **nBitSize**
[out] Size in bits of the found process variable

EC_T_INT **nBitOffs**
[out] Bit offset in the process data image

EC_T_BOOL **bIsInputData**
[out] Determines whether the found process variable is an input variable or an output variable

MAX_PROCESS_VAR_NAME_LEN
Maximum length of a process variable name: 71 characters

6.9.14 esGetSlaveInpVarInfoEx

EC_T_DWORD **esGetSlaveInpVarInfoEx** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wNumOfVarsToRead,
EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntriesEx,
EC_T_WORD *pwReadEntries
)
Gets the input process variable extended information entries of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number process variable entries that have been stored in pSlaveProcVarInfoEntries
- **pSlaveProcVarInfoEntriesEx** – [out] The read process variable extended information entries
- **pwReadEntries** – [out] The number of read process variable information entries

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

struct **EC_T_PROCESS_VAR_INFO_EX**

Public Members

EC_T_CHAR **szName**[MAX_PROCESS_VAR_NAME_LEN_EX]
[out] Name of the found process variable

EC_T_WORD **wDataType**
[out] Data type of the found process variable (according to ETG.1000, section 5). See also EcType.h, DEFTYPE_BOOLEAN.

EC_T_WORD **wFixedAddr**
[out] Station address of the slave that is owner of this variable

EC_T_INT **nBitSize**
[out] Size in bits of the found process variable

EC_T_INT **nBitOffs**
[out] Bit offset in the process data image

EC_T_BOOL **bIsInputData**
[out] Determines whether the found process variable is an input variable or an output variable

EC_T_WORD **wIndex**
[out] Object index

EC_T_WORD **wSubIndex**
[out] Object sub index

EC_T_WORD **wPdoIndex**
[out] Index of PDO (process data object)

EC_T_WORD **wWkcStateDiagOffs**
[out] Bit offset in the diagnostic image (API GetDiagnosisImagePtr)

EC_T_WORD **wMasterSyncUnit**
[out] Master Sync Unit ID (ENI: Slave/ProcessData/RxPdo[1..4]@Su, Slave/ProcessData/TxPdo[1..4]@Su, comment at Cyclic/Frame/Cmd)

EC_T_DWORD **dwTaskId**
[out] ID of task where process variable is located

EC_T_CYC_COPY_INFO **CopyInfo**
[out] Copy Info if applied to the variable

MAX_PROCESS_VAR_NAME_LEN_EX
Maximum length of an extended process variable name: 127 characters

6.9.15 esGetSlaveOutpVarInfo

EC_T_DWORD **esGetSlaveOutpVarInfo** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wNumOfVarsToRead,
EC_T_PROCESS_VAR_INFO *pSlaveProcVarInfoEntries,
EC_T_WORD *pwReadEntries
)
Gets the output process variable information entries of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)

- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number of found process variable entries
- **pSlaveProcVarInfoEntries** – [out] The read process variable information entries
- **pwReadEntries** – [out] The number of read process variable information entries

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO

6.9.16 esGetSlaveOutpVarInfoEx

EC_T_DWORD **esGetSlaveOutpVarInfoEx** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wNumOfVarsToRead,
EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntriesEx,
EC_T_WORD *pwReadEntries
)

Gets the output process variable extended information entries of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number of process variable information entries
- **pSlaveProcVarInfoEntriesEx** – [out] The read process extended variable entries
- **pwReadEntries** – [out] The number of read process variable information entries

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO_EX

6.9.17 esGetSlaveInpVarByObjectEx

EC_T_DWORD **esGetSlaveInpVarByObjectEx** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wIndex,
EC_T_WORD wSubIndex,
EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)

Gets the input process variable extended information entry by object index, subindex of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wIndex** – [in] Object index
- **wSubIndex** – [in] Object sub index
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

Returns

EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO_EX

6.9.18 esGetSlaveOutpVarByObjectEx

EC_T_DWORD **esGetSlaveOutpVarByObjectEx** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wIndex,
EC_T_WORD wSubIndex,
EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)

Gets the input process variable extended information entry by object index, subindex of a specific slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wIndex** – [in] Object index
- **wSubIndex** – [in] Object sub index
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

Returns

EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO_EX

6.9.19 esFindInpVarByName

EC_T_DWORD **esFindInpVarByName** (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)

Finds an input process variable information entry by the variable name.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable information entry

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO

6.9.20 esFindInpVarByNameEx

EC_T_DWORD **esFindInpVarByNameEx** (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)

Finds an input process variable extended information entry by the variable name.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO

6.9.21 esFindOutpVarByName

EC_T_DWORD **esFindOutpVarByName** (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)

Finds an output process variable information entry by the variable name.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)

- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable information entry

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO

6.9.22 esFindOutpVarByNameEx

EC_T_DWORD **esFindOutpVarByNameEx** (
EC_T_DWORD dwInstanceID,
const EC_T_CHAR *szVariableName,
EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)

Finds an output process variable extended information entry by the variable name.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

Returns
EC_E_NOERROR or error code

See also:

EC_T_PROCESS_VAR_INFO_EX

6.9.23 esWriteSlaveRegister

Warning: Changing contents of ESC registers may lead to unpredictable behavior of the SubDevices and/or the MainDevice.

EC_T_DWORD **esWriteSlaveRegister** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wRegisterOffset,
EC_T_BYTE *pbyData,
EC_T_WORD wLen,
EC_T_DWORD dwTimeout
)

Writes data into the ESC memory of a specified slave.

This function may not be called from within the JobTask's context

Warning: Changing contents of ESC registers may lead to unpredictable behavior of the slaves and/or the master.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wRegisterOffset** – [in] Register offset. E.g. use 0x0120 to write to the AL Control register.
- **pbyData** – [in] Buffer containing transfered data
- **wLen** – [in] Number of bytes to send
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes.

6.9.24 esReadSlaveRegister

```
EC_T_DWORD esReadSlaveRegister (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
```

Reads data from the ESC memory of a specified slave.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wRegisterOffset** – [in] Register offset. I.e. use 0x0130 to read the AL Status register.

- **pbyData** – [out] Buffer receiving transfered data
- **wLen** – [in] Number of bytes to receive
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if the slave is not present
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes.

6.9.25 esReadSlaveEEPRom

```
EC_T_DWORD esReadSlaveEEPRom (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPRomStartOffset,
    EC_T_WORD *pwReadData,
    EC_T_DWORD dwReadLen,
    EC_T_DWORD *pdwNumOutData,
    EC_T_DWORD dwTimeout
)
```

Read EEPROM data from a slave.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPRomStartOffset** – [in] Word address to start EEPROM read from
- **pwReadData** – [in] Pointer to EC_T_WORD array to carry the read data
- **dwReadLen** – [in] Size of the EC_T_WORD array provided at pwReadData (in EC_T_WORDs)
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD carrying actually read data (in EC_T_WORDs) after completion

- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Returns
EC_E_NOERROR or error code

6.9.26 esWriteSlaveEEPRom

EC_T_DWORD **esWriteSlaveEEPRom** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wEEPRomStartOffset,
const EC_T_WORD *pwWriteData,
EC_T_DWORD dwWriteLen,
EC_T_DWORD dwTimeout
)

Write EEPROM data to slave.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPRomStartOffset** – [in] Word address to start EEPROM Write from
- **pwWriteData** – [in] Pointer to WORD array carrying the data to write
- **dwWriteLen** – [in] Size of Write Data WORD array (in WORDS)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Returns
EC_E_NOERROR or error code

Note: The EEPROM's CRC at word 7 is re-calculated automatically if the written data is completely within the ESC Configuration Area in the EEPROM's SII.

6.9.27 esGetSimSlaveInfo

EC_T_DWORD **esGetSimSlaveInfo** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_SIM_SLAVE_INFO *pSimSlaveInfo
)

Get extended configuration and status information for slave from simulator.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID

- **wCfgFixedAddress** – [in] Slave's station address
- **pSimSlaveInfo** – [out] Extended configuration and status information for slave from simulator

Returns
EC_E_NOERROR or error code

Content of *EC_T_SIM_SLAVE_INFO*, *EC_T_SIM_SLAVE_CFG_INFO*, *EC_T_SIM_SLAVE_STATUS_INFO* is subject to be extended.

struct **EC_T_SIM_SLAVE_INFO**

Public Members

EC_T_SIM_SLAVE_CFG_INFO **oCfg**
[out] Config Info (ENI/EXI)

EC_T_SIM_SLAVE_STATUS_INFO **oStatus**
[out] Status Info

EC_T_DWORD **adwReserved**[32]
[out] Reserved

struct **EC_T_SIM_SLAVE_CFG_INFO**

Public Members

EC_T_WORD **wFixedAddress**
[out] Slave's station address (ENI/EXI: /EtherCATConfig/Config/Slave/Info/PhysAddr)

EC_T_BOOL **bPowerOff**
[out] Slave powered off on startup (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave@PowerOff)

EC_T_BOOL **bSimulated**
[out] Slave is simulated (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave/Simulated)

EC_T_BOOL **bIgnoreCoeDownloadError**
[out] Slave ignores CoE download errors (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave/Mailbox/CoE@IgnoreDownloadError)

EC_T_SIMULATOR_DEVICE_CONNECTION_DESC **aPortConnection**[4]
[out] Explicit port connection (optional) (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave/PortConnection)

EC_T_CHAR **szApplicationName**[EC_SIM_SLAVE_CFG_APP_NAME_SIZE]
[out] Configured slave application name (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave/Application/Name)

EC_T_CHAR **szApplicationParms**[EC_SIM_SLAVE_CFG_APP_PARMS_SIZE]
[out] Configured slave application parameters (EXI: /EtherCATConfig/ExtendedConfig/Slaves/Slave/Application/Parameter)

EC_T_DWORD **adwReserved**[16]
[out] Reserved

struct **EC_T_SIM_SLAVE_STATUS_INFO**

Public Members

EC_T_BOOL **bIsPowerOn**
[out] Slave is powered on. See esPowerSlave().

EC_T_BOOL **bIsPresent**
[out] Slave is present in topology segment and connected to the network. See esConnectPorts(), Hot Connect.

EC_T_WORD **wAlStatusReq**
[out] AL Status (0x0130) requested from Simulator application. See esSetSimSlaveState().

EC_T_WORD **wAlStatusCodeReq**
[out] AL Status (0x0134) requested from Simulator application. See esSetSimSlaveState().

EC_T_DWORD **adwRes1**[4]
[out] Reserved

EC_T_SIMULATOR_DEVICE_CONNECTION_DESC **aPortConnection**[4]
[out] See esConnectPorts()

EC_T_DWORD **adwRes2**[16]
[out] Reserved

6.9.28 esGetBusSlaveInfo

EC_T_DWORD **esGetBusSlaveInfo** (
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_BUS_SLAVE_INFO *pSlaveInfo
)
Return information about a slave connected to the EtherCAT bus.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveInfo** – [out] Information from the slave

Returns

- *EC_E_NOERROR* if successful

- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

Content of *EC_T_BUS_SLAVE_INFO* is subject to be extended.

struct **EC_T_BUS_SLAVE_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] The slave's ID to bind bus slave and config slave information

EC_T_DWORD **adwPortSlaveIds**[ESC_PORT_COUNT]
[out] The slave's ID of the slaves connected to ports. See Port slave ID's.

EC_T_WORD **wPortState**

[out] Port link state. Format: wwww xxxx yyyy zzzz (each nibble : port 3210)

wwww : Signal detected 1=yes, 0=no

xxxx : Loop closed 1=yes, 0=no

yyyy : Link established 1=yes, 0=no

zzzz : Slave connected 1=yes, 0=no (zzzz = logical result of w,x,y)

EC_T_WORD **wAutoIncAddress**
[out] The slave's auto increment address

EC_T_BOOL **bDcSupport**
[out] Slave supports DC (Bus Topology Scan)

EC_T_BOOL **bDc64Support**
[out] Slave supports 64 Bit DC (Bus Topology Scan)

EC_T_DWORD **dwVendorId**
[out] Vendor Identification stored in the EEPROM at offset 0x0008

EC_T_DWORD **dwProductCode**
[out] Product Code stored in the EEPROM at offset 0x000A

EC_T_DWORD **dwRevisionNumber**
[out] Revision number stored in the EEPROM at offset 0x000C

EC_T_DWORD **dwSerialNumber**
[out] Serial number stored in the EEPROM at offset 0x000E

EC_T_BYTE **byESCType**
[out] Type of ESC (Value of slave ESC register 0x0000)

EC_T_BYTE **byESCRevision**
[out] Revision number of ESC (Value of slave ESC register 0x0001)

EC_T_WORD **wESCBuild**
[out] Build number of ESC (Value of slave ESC register 0x0002)

EC_T_BYTE **byPortDescriptor**
[out] Port descriptor (Value of slave ESC register 0x0007)

EC_T_WORD **wFeaturesSupported**
[out] Features supported (Value of slave ESC register 0x0008)

EC_T_WORD **wStationAddress**
[out] The slave's station address (Value of slave ESC register 0x0010)

EC_T_WORD **wAliasAddress**
[out] The slave's alias address (Value of slave ESC register 0x0012)

EC_T_WORD **wAlStatus**
[out] AL status (Value of slave ESC register 0x0130)

EC_T_WORD **wAlStatusCode**
[out] AL status code. (Value of slave ESC register 0x0134 during last error acknowledge). This value is reset after a slave state change.

EC_T_DWORD **dwSystemTimeDifference**
[out] System time difference. (Value of slave ESC register 0x092C)

EC_T_WORD **wMbxSupportedProtocols**
[out] Supported Mailbox Protocols stored in the EEPROM at offset 0x001C

EC_T_WORD **wDlStatus**
[out] DL status (Value of slave ESC register 0x0110)

EC_T_WORD **wPrevPort**
[out] Connected port of the previous slave

EC_T_WORD **wIdentifyData**
[out] Last read identification value see *EC_T_CFG_SLAVE_INFO.wIdentifyAdo*

EC_T_BOOL **bLineCrossed**
[out] Line crossed was detected at this slave

EC_T_DWORD **dwSlaveDelay**
[out] Delay behind slave [ns].

EC_T_DWORD **dwPropagDelay**
[out] Propagation delay [ns]. ESC register 0x0928.

EC_T_BOOL **bIsRefClock**
[out] Slave is reference clock

EC_T_BOOL **bIsDeviceEmulation**
[out] Slave without Firmware. ESC register 0x0141, enabled by EEPROM offset 0x0000.8.

EC_T_WORD **wLineCrossedFlags**
[out] Combination of Line crossed flags

EC_T_DWORD **dwCyclicWkcErrorCnt**
[out] Counter for Cyclic WC Error

EC_T_DWORD **dwSlaveAbsentCnt**
[out] Counter for Absent/Not Present Slaves

EC_T_DWORD **dwUnexpectedStateCnt**
[out] Counter for Abnormal State Change

Get bus SubDevice info with station address example

```
        EC_T_BUS_SLAVE_INFO oBusSlaveInfo;

        /* get bus slave info of slave with station address 1001 */
        dwRes = esGetBusSlaveInfo(dwSimulatorId, EC_TRUE, 1001, &oBusSlaveInfo);
        if (dwRes != EC_E_NOERROR)
        {
            dwRetVal = dwRes;
            goto Exit;
        }
```

Get bus SubDevice info with auto-inc address example

```
        EC_T_BUS_SLAVE_INFO oBusSlaveInfo;

        /* get bus slave info of third slave (auto inc address 0, -1, -2, ...) */
        dwRes = esGetBusSlaveInfo(dwSimulatorId, EC_FALSE, (EC_T_WORD)-2, &
↪oBusSlaveInfo);
        if (dwRes != EC_E_NOERROR)
        {
            dwRetVal = dwRes;
            goto Exit;
        }
```

6.9.29 esReadSlaveIdentification

EC_T_DWORD **esReadSlaveIdentification**(
EC_T_DWORD dwInstanceID,
EC_T_BOOL bFixedAddressing,
EC_T_WORD wSlaveAddress,
EC_T_WORD wAdo,
EC_T_WORD *pwValue,
EC_T_DWORD dwTimeout
)

Read identification value from a slave.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing

- **wAdo** – [in] ADO used for identification command
- **pwValue** – [out] Pointer to Word value containing the Identification value
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_ADO_NOT_SUPPORTED* if the slave does not support requesting ID mechanism
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

6.9.30 esGetSlaveStatistics

EC_T_DWORD **esGetSlaveStatistics** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_SLVSTATISTICS_DESC *pSlaveStatisticsDesc
)

Get Slave's statistics counter.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave id
- **pSlaveStatisticsDesc** – [out] Pointer to structure EC_T_SLVSTATISTICS_DESC

Returns

EC_E_NOERROR or error code

6.9.31 esClearSlaveStatistics

EC_T_DWORD **esClearSlaveStatistics** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId
)

Clears all error registers of a slave.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave Id, INVALID_SLAVE_ID clears all slaves

Returns

EC_E_NOERROR or error code

6.10 ADS over EtherCAT® (AoE)

To handle AoE object transfers within the application, the callbacks pfnAoeRead, pfnAoeWrite and/or pfnAoeReadWrite can be registered using `esSetSlaveAoeObjectTransferCallbacks()` or `esSetSlaveSscApplication()`, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo.

See also *AoE Simulator and MainDevice Example*

6.10.1 esAoeGetSlaveNetId

EC_T_DWORD **esAoeGetSlaveNetId** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poAoeNetId
)

Retrieve the NetID of a specific EtherCAT device.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poAoeNetId** – [out] AoE NetID of the corresponding slave

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_ADS_IS_RUNNING* if ADS server is running

6.10.2 esAoeRead

EC_T_DWORD **esAoeRead** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)

Execute an AoE mailbox read request to an EtherCAT slave device.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE read command index group
- **dwIndexOffset** – [in] AoE read command index offset
- **dwDataLen** – [in] Buffer length [bytes]
- **pbyData** – [out] Buffer receiving transferred data
- **pdwDataOutLen** – [out] Number of bytes read from the target device
- **pdwErrorCode** – [out] AoE response error code
- **pdwCmdResult** – [out] AoE read command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

Returns

- *EC_E_NOERROR* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

6.10.3 esAoeWrite

EC_T_DWORD **esAoeWrite** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)

Execute an AoE mailbox write request to an EtherCAT slave device.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE write command index group
- **dwIndexOffset** – [in] AoE write command index offset
- **dwDataLen** – [in] Buffer length [bytes]
- **pbyData** – [in] Buffer containing transfered data
- **pdwErrorCode** – [out] Pointer to AoE response error code
- **pdwCmdResult** – [out] Pointer to AoE write command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

Returns

- *EC_E_NOERROR* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

6.10.4 esAoeReadWrite

EC_T_DWORD **esAoeReadWrite** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwReadDataLen,
    EC_T_DWORD dwWriteDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)

Execute an AoE mailbox read/write request to an EtherCAT slave device.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE read/write command index group
- **dwIndexOffset** – [in] AoE read/write command index offset
- **dwReadDataLen** – [in] Number of bytes to read from the target device
- **dwWriteDataLen** – [in] Number of bytes to write to the target device
- **pbyData** – [in, out] Buffer containing and receiving transfered data
- **pdwDataOutLen** – [out] Number of bytes read from the target device
- **pdwErrorCode** – [out] Pointer to AoE response error code
- **pdwCmdResult** – [out] Pointer to AoE write command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. EC_NOWAIT is not valid.

Returns

- *EC_E_NOERROR* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

6.10.5 esSetSlaveAoeObjectTransferCallbacks

EC_T_DWORD **esSetSlaveAoeObjectTransferCallbacks** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_PF_AOE_READ_CB pfRead,
EC_T_PF_AOE_WRITE_CB pfWrite,
EC_T_PF_AOE_READWRITE_CB pfReadWrite,
EC_T_VOID *pvContext
)

Set AoE read/write callbacks.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **pfRead** – [in] Read callback function
- **pfWrite** – [in] Write callback function
- **pfReadWrite** – [in] ReadWrite callback function
- **pvContext** – [in] Pointer to context passed as first parameters to callback functions

Returns

EC_E_NOERROR or error code

6.10.6 AoE Simulator and MainDevice Example

The following example demonstrates how to handle AoE object transfers within the simulator application:

AoE Simulator and MainDevice Example

The following example handlers can be registered at the EC-Simulator:

```
/* EC_T_PF_AOE_READ_CB */
EC_T_DWORD EC_FNCALL myAppAoeReadObjectCallback(
    EC_T_VOID* /* pvContext */, EC_T_DWORD dwSimulatorId, EC_T_WORD␣
↪wCfgFixedAddress,
    EC_T_AOE_NETID* /* poSenderNetId */, EC_T_WORD /* wSenderPort */, EC_T_AOE_
↪NETID* /* poTargetNetId */, EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup, EC_T_DWORD dwIndexOffset,
    EC_T_BYTE* pbyReadData, EC_T_DWORD* pdwReadDataLen,
    EC_T_DWORD* pdwErrorCode, EC_T_DWORD* pdwCmdResult)
{
    if (65535 != wTargetPort)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_TARGET_PORT_NOT_FOUND);
        goto Exit;
    }

    if (0xF302 /* ADSIGRP_CANOPEN_SDO */ != dwIndexGroup)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDGRP);
        goto Exit;
    }
```

(continues on next page)

(continued from previous page)

```
    /* check for object 0x2003, subindex 0, no complete access */
    if (0x20030000 != dwIndexOffset /* example value */)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_NOTFOUND);
        goto Exit;
    }
    /* check for size of object 0x2003, subindex 0 */
    if (4 != EC_GETDWORD(pdwReadDataLen) /* UDINT: sizeof(EC_T_DWORD) */)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDSIZE);
        goto Exit;
    }

    /* all checks passed, return data and set success */
    EC_SET_FRM_DWORD(pbyReadData, 0x12345678);
    EC_SETDWORD(pdwErrorCode, EC_E_NOERROR);

Exit:
    EcLogMsg(EC_LOG_LEVEL_VERBOSE, (pEcLogContext, EC_LOG_LEVEL_VERBOSE,
↪"myAppAoeReadObjectCallback(%d, %d, %d, 0x%04X, 0x%04X, %d bytes, 0x%08X, 0x
↪%08X): %s (0x%08X)\n",
        dwSimulatorId, wCfgFixedAddress, wTargetPort,
        dwIndexGroup, dwIndexOffset, EC_GETDWORD(pdwReadDataLen), EC_
↪GETDWORD(pdwErrorCode), EC_GETDWORD(pdwCmdResult),
        esGetText(dwSimulatorId, EC_GETDWORD(pdwErrorCode)), EC_
↪GETDWORD(pdwErrorCode)));
    return EC_GETDWORD(pdwErrorCode);
}

/* EC_T_PF_AOE_WRITE_CB */
EC_T_DWORD EC_FNCALL myAppAoeWriteObjectCallback(
    EC_T_VOID* /* pvContext */, EC_T_DWORD dwSimulatorId, EC_T_WORD␣
↪wCfgFixedAddress,
    EC_T_AOE_NETID* /* poSenderNetId */, EC_T_WORD /* wSenderPort */, EC_T_AOE_
↪NETID* /* poTargetNetId */, EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup, EC_T_DWORD dwIndexOffset,
    EC_T_BYTE* pbyWriteData, EC_T_DWORD dwWriteDataLen,
    EC_T_DWORD* pdwErrorCode, EC_T_DWORD* pdwCmdResult)
{
    EC_T_DWORD dwVal = 0;
    if (65535 != wTargetPort)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_TARGET_PORT_NOT_FOUND);
        goto Exit;
    }

    if (0xF302 /* ADSIGRP_CANOPEN_SDO */ != dwIndexGroup)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDGRP);
        goto Exit;
    }

    /* check for object 0x2003, subindex 0, no complete access */
    if (0x20030000 != dwIndexOffset /* example value */)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_NOTFOUND);
        goto Exit;
    }
    /* check for size of object 0x2003, subindex 0 */
    if (4 != dwWriteDataLen /* UDINT: sizeof(EC_T_DWORD) */)
    {
```

(continues on next page)

(continued from previous page)

```
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDSIZE);
        goto Exit;
    }

    /* all checks passed, get data and set success */
    dwVal = EC_GET_FRM_DWORD(pbyWriteData);
    EC_SETDWORD(pdwErrorCode, EC_E_NOERROR);
Exit:
    EcLogMsg(EC_LOG_LEVEL_VERBOSE, (pEcLogContext, EC_LOG_LEVEL_VERBOSE,
↪"myAppAoeWriteObjectCallback(%d, %d, %d, 0x%04X, 0x%04X, %d bytes, 0x%08X, 0x
↪%08X) = %d (0x%08X): %s (0x%08X)\n",
        dwSimulatorId, wCfgFixedAddress, wTargetPort,
        dwIndexGroup, dwIndexOffset, dwWriteDataLen, EC_GETDWORD(pdwErrorCode), EC_
↪GETDWORD(pdwCmdResult),
        dwVal, dwVal,
        esGetText(dwSimulatorId, EC_GETDWORD(pdwErrorCode)), EC_
↪GETDWORD(pdwErrorCode)));
    return EC_GETDWORD(pdwErrorCode);
}

/* EC_T_PF_AOE_READWRITE_CB */
EC_T_DWORD EC_FNCALL myAppAoeReadWriteObjectCallback(
    EC_T_VOID* /* pvContext */, EC_T_DWORD dwSimulatorId, EC_T_WORD␣
↪wCfgFixedAddress,
    EC_T_AOE_NETID* /* poSenderNetId */, EC_T_WORD /* wSenderPort */, EC_T_AOE_
↪NETID* /* poTargetNetId */, EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup, EC_T_DWORD dwIndexOffset,
    EC_T_BYTE* pbyReadData, EC_T_DWORD* pdwReadDataLen, EC_T_BYTE* pbyWriteData,␣
↪EC_T_DWORD dwWriteDataLen,
    EC_T_DWORD* pdwErrorCode, EC_T_DWORD* pdwCmdResult)
{
    EC_T_DWORD dwVal = 0;
    if (65535 != wTargetPort)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_TARGET_PORT_NOT_FOUND);
        goto Exit;
    }

    if (0xF302 /* ADSIGRP_CANOPEN_SDO */ != dwIndexGroup)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDGRP);
        goto Exit;
    }

    /* check for object 0x2003, subindex 0, no complete access */
    if (0x20030000 != dwIndexOffset /* example value */)
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_NOTFOUND);
        goto Exit;
    }
    /* check for size of object 0x2003, subindex 0 */
    if ((4 != EC_GETDWORD(pdwReadDataLen) /* UDINT: sizeof(EC_T_DWORD) */)
        || (4 != dwWriteDataLen /* UDINT: sizeof(EC_T_DWORD) */))
    {
        EC_SETDWORD(pdwErrorCode, EC_E_AOE_INVALIDSIZE);
        goto Exit;
    }

    /* all checks passed, set and get data and set success */
    EC_SET_FRM_DWORD(pbyReadData, 0x12345678);
    dwVal = EC_GET_FRM_DWORD(pbyWriteData);
```

(continues on next page)

(continued from previous page)

```
    EC_SETDWORD(pdwErrorCode, EC_E_NOERROR);

Exit:

    EcLogMsg(EC_LOG_LEVEL_VERBOSE, (pEcLogContext, EC_LOG_LEVEL_VERBOSE,
↪"myAppAoeReadWriteObjectCallback(%d, %d, %d, 0x%04X, 0x%04X, read %d bytes,␣
↪write %d bytes, 0x%08X, 0x%08X) = %d (0x%08X): %s (0x%08X)\n",
        dwSimulatorId, wCfgFixedAddress, wTargetPort,
        dwIndexGroup, dwIndexOffset, EC_GETDWORD(pdwReadDataLen), dwWriteDataLen,␣
↪EC_GETDWORD(pdwErrorCode), EC_GETDWORD(pdwCmdResult),
        dwVal, dwVal,
        esGetText(dwSimulatorId, EC_GETDWORD(pdwErrorCode)), EC_
↪GETDWORD(pdwErrorCode)));
    return EC_GETDWORD(pdwErrorCode);
}
```

The following example demonstrates object transfers between MainDevice and Simulator:

```
    EC_T_DWORD dwRes = EC_E_ERROR;
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_BYTE  abyWriteData[sizeof(EC_T_DWORD)];
    EC_T_BYTE  abyReadData[sizeof(EC_T_DWORD)];
    EC_T_BYTE  abyReadWriteData[sizeof(EC_T_DWORD)];
    EC_T_DWORD dwDataOutLen = 0;
    EC_T_AOE_NETID oAoeNetId;
    EC_T_DWORD dwSlaveId = emGetSlaveId(dwMasterId, wSlaveAddress);
    EC_T_DWORD dwErrorCode = EC_E_ERROR;
    EC_T_DWORD dwCmdResult = EC_E_ERROR;

    OsMemset(abyWriteData, 0, sizeof(abyWriteData));
    OsMemset(abyReadData, 0, sizeof(abyReadData));
    OsMemset(abyReadWriteData, 0, sizeof(abyReadWriteData));

    /* EC-Master: get configured AoE net ID */
    dwRes = emAoeGetSlaveNetId(dwMasterId, dwSlaveId, &oAoeNetId);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeGetSlaveNetId failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: register AoE handlers */
    dwRes = esSetSlaveAoeObjectTransferCallbacks(dwSimulatorId, wSlaveAddress,
        myAppAoeReadObjectCallback, myAppAoeWriteObjectCallback,␣
↪myAppAoeReadWriteObjectCallback, pAppContext);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_
↪ERROR, "esSetSlaveAoeObjectTransferCallbacks failed: %s (0x%lx))\n",␣
↪esGetText(dwSimulatorId, dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: set PREOP state */
    dwRes = emSetMasterState(dwMasterId, 30000, eEcatState_PREOP);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emSetMasterState failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
```

(continues on next page)

(continued from previous page)

```
        dwRetVal = dwRes;
        goto Exit;
    }

#define AOE_TARGET_PORT     65535
#define AOE_INDEX_GROUP     0xF302  /* ADSIGRP_CANOPEN_SDO */

    /* CoE objects to access via AoE */
#define AOE_COE_OBJ_IDX             0x2003
#define AOE_COE_OBJ_SUBINDEX        0
#define AOE_COE_OBJ_COMPLETEACCESS EC_FALSE

    /* Bit 16-31: index, Bit 8: complete access, Bit 0-7: subindex */
#define AOE_INDEX_OFFSET ((AOE_COE_OBJ_IDX << 16) | (AOE_COE_OBJ_COMPLETEACCESS <<␣
↪8) | AOE_COE_OBJ_SUBINDEX)

    /* EC-Master: write AoE to slave */
    EC_SET_FRM_DWORD(abyWriteData, 0x11223344 /* example value */);
    dwRes = emAoeWrite(dwMasterId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT, AOE_
↪INDEX_GROUP, AOE_INDEX_OFFSET,
        sizeof(EC_T_DWORD), abyWriteData, &dwErrorCode, &dwCmdResult, MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeWrite failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: read AoE from slave */
    EC_SET_FRM_DWORD(abyReadData, 0);
    dwRes = emAoeRead(dwMasterId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT, AOE_
↪INDEX_GROUP, AOE_INDEX_OFFSET,
        sizeof(EC_T_DWORD), abyReadData, &dwDataOutLen, &dwErrorCode, &dwCmdResult,
↪ MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeRead failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: read + write AoE from/to slave */
    EC_SET_FRM_DWORD(abyReadWriteData, 0x11223344 /* example value */);
    dwRes = emAoeReadWrite(dwMasterId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT, AOE_
↪INDEX_GROUP, AOE_INDEX_OFFSET,
        sizeof(EC_T_DWORD), sizeof(EC_T_DWORD), abyReadWriteData, &dwDataOutLen, &
↪dwErrorCode, &dwCmdResult, MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeReadWrite failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: write AoE to slave */
    EC_SET_FRM_DWORD(abyWriteData, 0x11223344 /* example value */);
    dwRes = esAoeWrite(dwSimulatorId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT, AOE_
↪INDEX_GROUP, AOE_INDEX_OFFSET,
```

(continues on next page)

(continued from previous page)

```
        sizeof(EC_T_DWORD), abyWriteData, &dwErrorCode, &dwCmdResult, MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeWrite failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: read AoE from slave */
    EC_SET_FRM_DWORD(abyReadData, 0);
    dwRes = esAoeRead(dwSimulatorId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT, AOE_
↪INDEX_GROUP, AOE_INDEX_OFFSET,
        sizeof(EC_T_DWORD), abyReadData, &dwDataOutLen, &dwErrorCode, &dwCmdResult,
↪ MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeRead failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: read + write AoE from/to slave */
    EC_SET_FRM_DWORD(abyReadWriteData, 0x11223344 /* example value */);
    dwRes = esAoeReadWrite(dwSimulatorId, dwSlaveId, &oAoeNetId, AOE_TARGET_PORT,␣
↪AOE_INDEX_GROUP, AOE_INDEX_OFFSET,
        sizeof(EC_T_DWORD), sizeof(EC_T_DWORD), abyReadWriteData, &dwDataOutLen, &
↪dwErrorCode, &dwCmdResult, MBX_TIMEOUT);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emAoeReadWrite failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }
```

6.11 CAN application protocol over EtherCAT® (CoE)

To handle CoE object transfers within the application, the callbacks pfnCoeRead and/or pfnCoeWrite can be registered using *esSetSlaveCoeObjectTransferCallbacks()* or *esSetSlaveSscApplication()*, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo.

See also *CoE transfer Simulator and MainDevice Example*

6.11.1 esExtendSlaveCoeObjectDictionary

EC_T_DWORD **esExtendSlaveCoeObjectDictionary**(
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    EC_T_COE_DICTIONARY_DESC *pDict
)

Add data types and / or objects to slave's CoE object dictionary.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **pDict** – [in] Pointer to dictionary descriptor

Returns
EC_E_NOERROR or error code

6.11.2 esDeleteSlaveCoeObject

EC_T_DWORD **esDeleteSlaveCoeObject** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_WORD wObjectIndex
)

Delete object from slave's CoE object dictionary.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address
- **wObjectIndex** – [in] Object index

Returns
EC_E_NOERROR or error code

6.11.3 esClearSlaveCoeObjectDictionary

EC_T_DWORD **esClearSlaveCoeObjectDictionary** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress
)

Delete all objects from slave's CoE object dictionary.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address

Returns
EC_E_NOERROR or error code

6.11.4 esResetSlaveCoeObjectDictionary

EC_T_DWORD **esResetSlaveCoeObjectDictionary** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress
)

Reset all objects from slave's CoE object dictionary to default.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address

Returns
EC_E_NOERROR or error code

6.11.5 esSetSlaveCoeObjectTransferCallbacks

EC_T_DWORD **esSetSlaveCoeObjectTransferCallbacks** (
EC_T_DWORD dwInstanceId,
EC_T_WORD wCfgFixedAddress,
EC_T_WORD wObjectIndex,
EC_T_PF_COE_READ_CB pfRead,
EC_T_PF_COE_WRITE_CB pfWrite,
EC_T_VOID *pvContext
)

Set SDO upload / download transfer callbacks.

Note: pfRead / pfWrite can also be registered if object dictionary unavailable at EC-Simulator. *esClearSlaveCoeObjectDictionary()* has to be called first. See *esClearSlaveCoeObjectDictionary()*. In case of using an ENI and the simulator's default dictionary wObjectIndex must be set to 0xFFFF.

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave's station address. 0: all slaves.
- **wObjectIndex** – [in] Object index. 0xffff: all objects
- **pfRead** – [in] Upload callback function
- **pfWrite** – [in] Download callback function
- **pvContext** – [in] Pointer to context passed as first parameters to callback functions
- **dwInstanceId** – **[in]** Simulator Instance ID
- **wCfgFixedAddress** – **[in]** Slave's station address. 0: All slaves.
- **wObjectIndex** – **[in]** Object index. 0xFFFF: all objects.
- **pfRead** – **[in]** Upload callback function. EC_NULL: clears callback if already registered.
- **pfWrite** – **[in]** Download callback function. EC_NULL: clears callback if already registered.

- **pvContext** – **[in]** Arbitrarily application-defined parameter passed to transfer callback functions

Returns

- *EC_E_NOERROR*: Success
- *EC_E_INVALIDPARM*: if dwInstanceId is out of range
- *EC_E_INVALIDSTATE*: if simulator instance (dwInstanceId) not initialized, see *esInitSimulator()*
- *EC_E_NOTFOUND*: if slave not found in ENI/EXI (wCfgFixedAddress)
- *EC_E_NO_MBX_SUPPORT*: if slave without Mailbox support
- *EC_E_SDO_ABORTCODE_INDEX*: if nonexistent object (wObjectIndex)

See also:

- *esSetSlaveSscApplication()*
- *esClearSlaveCoeObjectDictionary()*

typedef EC_T_VOID (***EC_T_PF_COE_READ_CB**)(EC_T_VOID *pvContext, EC_T_DWORD dwSimulatorId, EC_T_WORD wCfgFixedAddress, EC_T_WORD wIndex, EC_T_BYTE bySubindex, EC_T_DWORD dwSize, EC_T_BYTE *pbyData, EC_T_DWORD *pdwOutDataLen, EC_T_BOOL bCompleteAccess, EC_T_BOOL bCheckReadAccess, EC_T_DWORD *pdwErrorCode)

CoE Read Object Callback: Handle read requests from the master in the simulator application. The simulator has pre-handled the request and sets pdwErrorCode accordingly.

Param pvContext
[in] Arbitrarily application-defined parameter passed to callback

Param dwSimulatorId
[in] Simulator Instance ID

Param wCfgFixedAddress
[in] Slave fixed address from master request

Param wIndex
[in] Object Index from master request, e.g. 0x1018

Param bySubindex
[in] Object SubIndex from master request

Param dwSize
[in] Read buffer size according to MbxIn Sync Manager size

Param pbyData
[inout] Read data buffer: pre-filled by the simulator if the object exists in the simulated slave's object dictionary and is read-able in the current EtherCAT state

Param pdwOutDataLen
[inout] Read data length: pre-filled by the simulator if the object exists in the simulated slave's object dictionary and is read-able in the current EtherCAT state, else the simulator application shall set the value accordingly

Param bCompleteAccess
[in] EC_TRUE: CoE SDO Upload with Complete Access. See also EC_MAILBOX_FLAG_SDO_COMPLETE.

Param bCheckReadAccess
[in] EC_TRUE: The simulator application should check the read access, e.g. if the object does not exist in the simulated slave's object dictionary

Param pdwErrorCode
[inout] Current transfer error code pre-filled by the simulator. The simulator application shall set the current transfer error accordingly within the callback.

- *EC_E_NOERROR*: if object exists in the simulated slave's object dictionary and is read-able in the current EtherCAT state
- *EC_E_SDO_ABORTCODE_INDEX*: if object does not exist in the simulated slave's object dictionary and is read-able in the current EtherCAT state
- *EC_E_SDO_ABORTCODE_OFFSET*: if object SubIndex does not exist in the accessed object in the simulated slave's object dictionary
- *EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH*: if the read data length does not match the accessed object length
- *EC_E_SDO_ABORTCODE_ACCESS*: if accessed Entry does not support commanded accesstype e.g. complete access is not supported on simple objects or ENUM descriptions
- *EC_E_SDO_ABORTCODE_WRITEONLY*: if accessed Entry does not support read operation
- *EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE*: if entry cannot be read in the current EtherCAT state
- EC_E_…

typedef EC_T_VOID (***EC_T_PF_COE_WRITE_CB**)(EC_T_VOID *pvContext, EC_T_DWORD dwSimulatorId, EC_T_WORD wCfgFixedAddress, EC_T_WORD wIndex, EC_T_BYTE bySubindex, EC_T_DWORD dwSize, EC_T_BYTE *pbyData, EC_T_BOOL bCompleteAccess, EC_T_BOOL bCheckWriteAccess, EC_T_DWORD *pdwErrorCode)

CoE Write Object Callback: Handle write requests from the master in the simulator application. The simulator has pre-handled the request and sets pdwErrorCode accordingly.

Param pvContext
[in] Arbitrarily application-defined parameter passed to callback

Param dwSimulatorId
[in] Simulator Instance ID

Param wCfgFixedAddress
[in] Slave fixed address from master request

Param wIndex
[in] Object Index from master request, e.g. 0x40A2

Param bySubindex
[in] Object SubIndex from master request

Param dwSize
[in] Write data length

Param pbyData
[inout] Write data buffer: pre-filled by the simulator if the object exists in the simulated slave's object dictionary and is write-able in the current EtherCAT state

Param bCompleteAccess
[in] EC_TRUE: CoE SDO Download with Complete Access. See also EC_MAILBOX_FLAG_SDO_COMPLETE.

Param bCheckWriteAccess
[in] EC_TRUE: The simulator application should check the write access, e.g. if the object does not exist in the simulated slave's object dictionary

Param pdwErrorCode
[inout] Current transfer error code pre-filled by the simulator. The simulator application shall set the current transfer error accordingly within the callback.

- *EC_E_NOERROR*: if object exists in the simulated slave's object dictionary and is write-able in the current EtherCAT state
- *EC_E_SDO_ABORTCODE_INDEX*: if object does not exist in the simulated slave's object dictionary and is write-able in the current EtherCAT state
- *EC_E_SDO_ABORTCODE_OFFSET*: if object SubIndex does not exist in the accessed object in the simulated slave's object dictionary
- *EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH*: if the write data length does not match the accessed object length
- *EC_E_SDO_ABORTCODE_ACCESS*: if accessed Entry does not support commanded accesstype e.g. complete access is not supported on simple objects or ENUM descriptions
- *EC_E_SDO_ABORTCODE_READONLY*: if accessed Entry does not support write operation
- *EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE*: if entry cannot be write in the current EtherCAT state
- EC_E_…

6.11.6 esCoeSdoDownload

EC_T_DWORD **esCoeSdoDownload** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_BYTE byObSubIndex,
const EC_T_BYTE *pbyData,
EC_T_DWORD dwDataLen,
EC_T_DWORD dwTimeout,
EC_T_DWORD dwFlags
)

Execute a CoE SDO download to an EtherCAT slave device.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object Index
- **byObSubIndex** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed.
- **pbyData** – [in] Buffer containing transferred data. Must not be EC_NULL if dwDataLen > 0.
- **dwDataLen** – [in] Buffer length [bytes]. Can be 0 for example for array.
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized

- *EC_E_INVALIDPARM* if dwInstanceID is out of range, pbyData is EC_NULL and dwDataLen > 0, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

6.11.7 esCoeSdoUpload

EC_T_DWORD **esCoeSdoUpload** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_BYTE byObSubIndex,
EC_T_BYTE *pbyData,
EC_T_DWORD dwDataLen,
EC_T_DWORD *pdwOutDataLen,
EC_T_DWORD dwTimeout,
EC_T_DWORD dwFlags
)

Execute a CoE SDO upload from an EtherCAT slave device to the master.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub index. If Complete Access only 0 or 1 allowed.
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **pdwOutDataLen** – [out] Length of received data [byte]
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT

- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

6.11.8 esCoeGetODList

EC_T_DWORD **esCoeGetODList** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_COE_ODLIST_TYPE eListType,
EC_T_BYTE *pbyData,
EC_T_DWORD dwDataLen,
EC_T_COE_ODLIST *poOdList,
EC_T_DWORD dwTimeout
)

Gets a list of object IDs that are available in a slave.

This function may not be called from within the JobTask's context.

Note: The data buffer will receive the slave response containing the list type followed by the list itself. Therefore the buffer must be 2 bytes bigger than the expected list size.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **eListType** – [in] Which object types shall be transferred
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **poOdList** – [out] Received OD list object
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present

- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

6.11.9 esCoeGetODListReq

EC_T_DWORD **esCoeGetODListReq** (
EC_T_DWORD dwInstanceID,
EC_T_MBXTFER *pMbxTfer,
EC_T_DWORD dwSlaveId,
EC_T_COE_ODLIST_TYPE eListType,
EC_T_DWORD dwTimeout
)

Initiates retrieval of a list of object IDs that are available in a slave and returns immediately.

A unique transfer ID must be written into EC_T_MBXTFER.dwTferId. EC_NOTIFY_MBOXRCV is given on completion.

Note: The mailbox transfer object will receive the slave response containing the list type followed by the list itself. Therefore the buffer must be 2 bytes bigger than the expected list size.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer
- **dwSlaveId** – [in] Slave ID
- **eListType** – [in] Which object types shall be transferred
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

6.11.10 esCoeGetObjectDesc

EC_T_DWORD **esCoeGetObjectDesc** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_CHAR *pchObName,
EC_T_WORD wObNameLen,
EC_T_COE_OBDESC *poObDesc,
EC_T_DWORD dwTimeout
)

Determines the description of a specific object.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **pchObName** – [out] Buffer receiving object name
- **wObNameLen** – [in] Buffer length
- **poObDesc** – [out] Received object description object
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

struct **EC_T_COE_OBDESC**

Public Members

EC_T_WORD **wObIndex**
Index in the object dictionary

EC_T_WORD **wDataType**
Data type of the object

EC_T_BYTE **byObjCode**
Object code, see Table 62, ETG.1000 section 6

EC_T_BYTE **byObjCategory**
Object category

EC_T_BYTE **byMaxNumSubIndex**
Maximum sub index number

EC_T_WORD **wObNameLen**
Length of the object name

EC_T_WORD **wStationAddress**
Station address of the slave

EC_T_CHAR ***pchObName**
Object name (not NULL terminated!)

6.11.11 esCoeGetObjectDescReq

EC_T_DWORD **esCoeGetObjectDescReq** (
EC_T_DWORD dwInstanceID,
EC_T_MBXTFER *pMbxTfer,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_DWORD dwTimeout
)

Initiates retrieval of the description of a specific object and returns immediately.

A unique transfer ID must be written into EC_T_MBXTFER.dwTferId. EC_NOTIFY_MBOXRCV is given on completion.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized

- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

6.11.12 esCoeGetEntryDesc

EC_T_DWORD **esCoeGetEntryDesc** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_BYTE byObSubIndex,
EC_T_BYTE byValueInfo,
EC_T_BYTE *pbyData,
EC_T_WORD wDataLen,
EC_T_COE_ENTRYDESC *poEntryDesc,
EC_T_DWORD dwTimeout
)

Determines the description of a specific object entry.

This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub-index
- **byValueInfo** – [in] The value info bit mask includes which elements shall be in the response. See Value info flags for available values.
- **pbyData** – [out] Buffer receiving additional entry desc data
- **wDataLen** – [in] Buffer length in bytes
- **poEntryDesc** – [out] Received entry desc object
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT

- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

struct **EC_T_COE_ENTRYDESC**

Public Members

EC_T_WORD **wObIndex**
Index in the object dictionary

EC_T_BYTE **byObSubIndex**
Sub index in the object dictionary

EC_T_BYTE **byValueInfo**
Bit mask which information is included in pbyData. See Value info flags.

EC_T_WORD **wDataType**
Object data type according to ETG.1000

EC_T_WORD **wBitLen**
Object size (number of bits)

EC_T_BYTE **byObAccess**
Access rights. See Object access flags.

EC_T_BOOL **bRxPdoMapping**
Object is mappable in a RxPDO

EC_T_BOOL **bTxPdoMapping**
Object is mappable in a TxPDO

EC_T_BOOL **bObCanBeUsedForBackup**
Object can be used for backup

EC_T_BOOL **bObCanBeUsedForSettings**
Object can be used for settings

EC_T_WORD **wStationAddress**
Station address of the slave

EC_T_WORD **wDataLen**
Size of the remaining object data

EC_T_BYTE ***pbyData**

Remaining object data: dwUnitType, pbyDefaultValue, pbyMinValue, pbyMaxValue, pbyDescription

(see ETG.1000.5 and ETG.1000.6)

See szUnitType, szDefaultValue, szMinValue, szMaxValue, szDescription in CoeReadObjectDictionary() in EcSdoServices.cpp as an example for evaluating EC_T_COE_ENTRYDESC.pbyData.

6.11.13 esCoeGetEntryDescReq

EC_T_DWORD **esCoeGetEntryDescReq** (
EC_T_DWORD dwInstanceID,
EC_T_MBXTFER *pMbxTfer,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_BYTE byObSubIndex,
EC_T_BYTE byValueInfo,
EC_T_DWORD dwTimeout
)

Initiates retrieval of the description of a specific object entry and returns immediately.

A unique transfer ID must be written into EC_T_MBXTFER.dwTferId. EC_NOTIFY_MBOXRCV is given on completion.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub-index
- **byValueInfo** – [in] The value info bit mask includes which elements shall be in the response. See Value info flags for available values.
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running

- *CoE SDO error code*

6.11.14 CoE transfer Simulator and MainDevice Example

The following example demonstrates how to handle CoE object transfers within the simulator application:

CoE transfer Simulator and MainDevice Example

The following example handlers can be registered at the EC-Simulator:

```
/* Length check in application if Object Dictionary unavailable at EC-Simulator */
static EC_T_DWORD myAppCoeObjectGetDataLen(
    EC_T_VOID* /* pvContext */, EC_T_WORD /* wCfgFixedAddress */,
    EC_T_WORD wObjectIndex, EC_T_BYTE bySubindex,
    EC_T_BOOL /* bCompleteAccess */, EC_T_DWORD* pdwDataLen)
{
    EC_T_DWORD dwRetVal = EC_E_SDO_ABORTCODE_INDEX;

    /* example for 0x1018 Identity Object, SubIndex 1 Vendor ID */
    if ((0x1018 == wObjectIndex) && (1 == bySubindex))
    {
        *pdwDataLen = 4;
        dwRetVal = EC_E_NOERROR;
    }

    return dwRetVal;
}

/* EC_T_PF_COE_READ_CB */
static EC_T_VOID EC_FNCALL myAppCoeReadObjectCallback(
    EC_T_VOID* pvContext, EC_T_DWORD /* dwSimulatorId */, EC_T_WORD␣
↪wCfgFixedAddress,
    EC_T_WORD wObjectIndex, EC_T_BYTE bySubindex,
    EC_T_DWORD dwBufSize, EC_T_BYTE* pbyData, EC_T_DWORD* pdwOutDataLen,
    EC_T_BOOL bCompleteAccess,
    EC_T_BOOL bCheckReadAccess /* EC_TRUE: Object Dictionary unavailable at EC-
↪Simulator */,
    EC_T_DWORD* pdwErrorCode /* see EC_E_SDO_ABORTCODE_... */)
{
    EC_T_DWORD dwDataLen = *pdwOutDataLen;

    /* check transfer parameters in application if Object Dictionary unavailable␣
↪at EC-Simulator */
    if (bCheckReadAccess)
    {
        *pdwErrorCode = myAppCoeObjectGetDataLen(pvContext, wCfgFixedAddress,␣
↪wObjectIndex, bySubindex, bCompleteAccess, &dwDataLen);
        if ((EC_E_NOERROR == *pdwErrorCode) && (dwBufSize < dwDataLen))
        {
            *pdwErrorCode = EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_LOW;
        }
    }
    if (EC_E_NOERROR == *pdwErrorCode)
    {
        /* provide data. example for 0x1018 Identity Object, SubIndex 1 Vendor ID␣
↪*/
        if ((0x1018 == wObjectIndex) && (1 == bySubindex))
        {
            OsDbgAssert(4 == dwDataLen); EC_SET_FRM_DWORD((EC_T_DWORD*)pbyData,␣
↪0x12345678);
```

(continues on next page)

(continued from previous page)

```
        }
    }
    else
    {
        dwDataLen = 0;
    }

    *pdwOutDataLen = dwDataLen;
    return;
}

/* EC_T_PF_COE_WRITE_CB */
static EC_T_VOID EC_FNCALL myAppCoeWriteObjectCallback(
    EC_T_VOID* /* pvContext */, EC_T_DWORD /* dwSimulatorId */, EC_T_WORD /*␣
↪wCfgFixedAddress */,
    EC_T_WORD wObjectIndex, EC_T_BYTE bySubindex,
    EC_T_DWORD dwSize, EC_T_BYTE* pbyData,
    EC_T_BOOL /* bCompleteAccess */,
    EC_T_BOOL bCheckWriteAccess /* EC_TRUE if Object Dictionary unavailable at EC-
↪Simulator */,
    EC_T_DWORD* pdwErrorCode)
{
    OsDbgAssert(!bCheckWriteAccess /* Object Dictionary available */
                || (EC_E_SDO_ABORTCODE_INDEX == *pdwErrorCode));

    if ((0x2000 == wObjectIndex) && (0 == bySubindex))
    {
        /* check write access if Object Dictionary unavailable at EC-Simulator */
        if (bCheckWriteAccess && (2 != dwSize))
        {
            if (dwSize > 2)
            {
                *pdwErrorCode = EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_HIGH;
            }
            else
            {
                *pdwErrorCode = EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_LOW;
            }
            return;
        }
        /* handle data, e.g. myCoeSdoWrite(wObjectIndex, bySubindex, pbyData); */
        EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
↪"myAppCoeWriteObjectCallback: 0x%04X, %d: 0x%04X)\n", wObjectIndex, bySubindex,
↪ EC_GET_FRM_WORD(pbyData)));

        *pdwErrorCode = EC_E_NOERROR;
        return;
    }
}
```

The following example demonstrates objects transfers between MainDevice and Simulator:

```
    EC_T_DWORD dwRes = EC_E_ERROR;
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_BYTE  abyWriteData[sizeof(EC_T_WORD)];
    EC_T_BYTE  abyReadData[sizeof(EC_T_DWORD)];
    EC_T_DWORD dwDataOutLen = 0;

    OsMemset(abyWriteData, 0, sizeof(abyWriteData));
    OsMemset(abyReadData, 0, sizeof(abyReadData));
```

(continues on next page)

(continued from previous page)

```
    /* EC-Simulator: clear Object Dictionary */
    dwRes = esClearSlaveCoeObjectDictionary(dwSimulatorId, wSlaveAddress);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"esClearSlaveCoeObjectDictionary failed: %s (0x%lx))\n", esGetText(dwSimulatorId,
↪ dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: register CoE handlers */
    dwRes = esSetSlaveCoeObjectTransferCallbacks(dwSimulatorId, wSlaveAddress,␣
↪0xFFFF /* all objects */,
                                                 myAppCoeReadObjectCallback,␣
↪myAppCoeWriteObjectCallback, pAppContext);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_
↪ERROR, "esSetSlaveCoeObjectTransferCallbacks failed: %s (0x%lx))\n",␣
↪esGetText(dwSimulatorId, dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: set PREOP state */
    dwRes = emSetMasterState(dwMasterId, 30000, eEcatState_PREOP);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emSetMasterState failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: write CoE to slave */
    EC_SET_FRM_WORD(abyWriteData, 0x1234 /* example value */);
    dwRes = emCoeSdoDownload(dwMasterId, emGetSlaveId(dwMasterId, wSlaveAddress),␣
↪0x2000, 0, abyWriteData, sizeof(EC_T_WORD), 5000 /* timeout */, 0);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emCoeSdoDownload failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: read CoE from slave */
    EC_SET_FRM_DWORD(abyReadData, 0);
    dwRes = emCoeSdoUpload(dwMasterId, emGetSlaveId(dwMasterId, wSlaveAddress),␣
↪0x1018, 1 /* vendor id */, abyReadData, sizeof(EC_T_DWORD), &dwDataOutLen, 5000 /
↪* timeout */, 0);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emCoeSdoUpload failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: write CoE to slave */
```

(continues on next page)

(continued from previous page)

```
    EC_SET_FRM_WORD(abyWriteData, 0x1234 /* example value */);
    dwRes = esCoeSdoDownload(dwSimulatorId, esGetSlaveId(dwSimulatorId,␣
↪wSlaveAddress), 0x2000, 0, abyWriteData, sizeof(EC_T_WORD), 5000 /* timeout */,
↪ 0);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"esCoeSdoDownload failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Simulator: read CoE from slave */
    EC_SET_FRM_DWORD(abyReadData, 0);
    dwRes = esCoeSdoUpload(dwSimulatorId, esGetSlaveId(dwSimulatorId,␣
↪wSlaveAddress), 0x1018, 1 /* vendor id */, abyReadData, sizeof(EC_T_DWORD), &
↪dwDataOutLen, 5000 /* timeout */, 0);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emCoeSdoUpload failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }
```

6.12 Ethernet over EtherCAT® (EoE)

To handle EoE frames within the application, the callback pfnEoeReceive must be registered using *esSetSlaveSscApplication()*, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo.

See also *EoE Ping Example*

6.12.1 esEoeSendFrame

EC_T_DWORD **esEoeSendFrame** (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    EC_T_BYTE *pbyFrame,
    EC_T_DWORD dwFrameLen
)

Send EoE frame (queued)

–

Returns

EC_E_NOERROR or error code

6.12.2 esGetCfgSlaveEoeInfo

EC_T_DWORD **esGetCfgSlaveEoeInfo** (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_EOE_INFO *pSlaveEoeInfo
)

Return EoE information about a configured slave from the ENI file.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveEoeInfo** – [out] Information about the slave

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_NO_MBX_SUPPORT* if the slave does not support mailbox communication
- *EC_E_NO_EOE_SUPPORT* if the slave supports mailbox communication, but not EoE

Content of *EC_T_CFG_SLAVE_EOE_INFO* is subject to be extended.

struct **EC_T_CFG_SLAVE_EOE_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] Slave ID

EC_T_BOOL **bMacAddr**
[out] Indicates whether the MAC address could be read and is valid

EC_T_BYTE **abyMacAddr**[6]
[out] MAC address

EC_T_BOOL **bIpAddr**
[out] Indicates whether the IP address could be read and is valid

EC_T_IPADDR **oIpAddr**
[out] IP address

EC_T_BOOL **bSubnetMask**
[out] Indicates whether the subnet mask could be read and is valid

EC_T_IPADDR **oSubnetMask**
[out] Subnet mask

EC_T_BOOL **bDefaultGateway**
[out] Indicates whether the default gateway could be read and is valid

EC_T_IPADDR **oDefaultGateway**
[out] Default gateway

EC_T_BOOL **bDnsServer**
[out] Indicates whether the DNS server could be read and is valid

EC_T_IPADDR **oDnsServer**
[out] DNS server

EC_T_BOOL **bDnsName**
[out] Indicates whether the DNS name could be read and is valid

EC_T_CHAR **szDnsName**[32]
[out] DNS name

EC_T_BOOL **bDisableEoe**
[out] Indicates whether the EoE is Disabled or not

esGetCfgSlaveEoeInfo() Example

```
    EC_T_CFG_SLAVE_EOE_INFO oInfo;
    OsMemset(&oInfo, 0, sizeof(EC_T_CFG_SLAVE_EOE_INFO));

    dwRes = esGetCfgSlaveEoeInfo(pAppContext->dwInstanceId, EC_TRUE, wSlaveAddress,
↪ &oInfo);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"esGetCfgSlaveEoeInfo failed: %s (0x%lx))\n",
            esGetText(pAppContext->dwInstanceId, dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
↪"esGetCfgSlaveEoeInfo(%d): "
        "MAC address: %02X:%02X:%02X:%02X:%02X:%02X, IP address: %d.%d.%d.%d,␣
↪subnet mask: %d.%d.%d.%d, "
        "default gateway: %d.%d.%d.%d, DNS server: %d.%d.%d.%d, DNS name: %s\n",␣
↪wSlaveAddress,
        oInfo.abyMacAddr[0], oInfo.abyMacAddr[1], oInfo.abyMacAddr[2],
        oInfo.abyMacAddr[3], oInfo.abyMacAddr[4], oInfo.abyMacAddr[5],

        oInfo.oIpAddr.sAddr.by[0], oInfo.oIpAddr.sAddr.by[1], oInfo.oIpAddr.sAddr.
↪by[2], oInfo.oIpAddr.sAddr.by[3],
        oInfo.oSubnetMask.sAddr.by[0], oInfo.oSubnetMask.sAddr.by[1], oInfo.
↪oSubnetMask.sAddr.by[2], oInfo.oSubnetMask.sAddr.by[3],

        oInfo.oDefaultGateway.sAddr.by[0], oInfo.oDefaultGateway.sAddr.by[1],␣
↪oInfo.oDefaultGateway.sAddr.by[2], oInfo.oDefaultGateway.sAddr.by[3],
        oInfo.oDnsServer.sAddr.by[0], oInfo.oDnsServer.sAddr.by[1], oInfo.
↪oDnsServer.sAddr.by[2], oInfo.oDnsServer.sAddr.by[3], oInfo.szDnsName));
```

6.12.3 EoE Ping Example

The following example demonstrates how to customize EoE simulation using *esSetSlaveSscApplication()* and *esEoeSendFrame()*.

EoE Ping Example

The following code demonstrates how to receive EoE frames from the MainDevice and send answers back.

```
/*-----------------------------------------------------------------------------*/
/** \brief  EoE ARP request and PING request context structure
*/
typedef struct _T_MY_EOE_CONTEXT
{
    T_EC_DEMO_APP_CONTEXT* pAppContext;
    EC_T_WORD              wSlaveAddress;
    ETHERNET_ADDRESS       oMacAddress;
    EC_T_IPADDR            oIpV4Address;
    EC_T_LINK_FRAMEDESC    oSendFrame;
} T_MY_EOE_CONTEXT;

/*-----------------------------------------------------------------------------*/
/** \brief  EoE ARP request and PING request handler
*
* \return  pReplyFrameDesc->dwSize > 0 if handled with reply
*/
static EC_T_VOID myAppProcessEoeFrameArpAndPing(EC_T_LINK_FRAMEDESC*␣
↪pRequestFrameDesc, EC_T_LINK_FRAMEDESC* pReplyFrameDesc, ETHERNET_ADDRESS␣
↪MacAddress, EC_T_IPADDR IpV4Address)
{
    ETHERNET_FRAME* pRequest = (ETHERNET_FRAME*)pRequestFrameDesc->pbyFrame;
    ETHERNET_FRAME* pReply = (ETHERNET_FRAME*)pReplyFrameDesc->pbyFrame;

    /* prepare reply */
    OsMemcpy(pReply, pRequest, pRequestFrameDesc->dwSize);
    pReply->Destination = pRequest->Source;
    pReply->Source = MacAddress;
    pReplyFrameDesc->dwSize = 0;

    /* handle ARP / ping */
    switch (EC_ETHFRM_GET_FRAMETYPE(pRequest))
    {
    case ETHERNET_FRAME_TYPE_ARP:
    {
        EC_ARP_IP_HEADER* pArpRequest = (EC_ARP_IP_HEADER*)EC_ENDOF(pRequest); /*␣
↪skip ETHERNET_FRAME header */
        EC_ARP_IP_HEADER* pArpReply = (EC_ARP_IP_HEADER*)EC_ENDOF(pReply); /* skip␣
↪ETHERNET_FRAME header */

        /* only Ethernet MAC and IPv4 ARP requests supported */
        if ((EC_NTOHS(pArpRequest->wHwAddressType) == EC_ARP_HW_TYPE_ETHERNET)
            && (EC_NTOHS(pArpRequest->wProtocolType) == ETHERNET_FRAME_TYPE_IP)
            && (pArpRequest->byHwAddressLength == ETHERNET_ADDRESS_LEN)
            && (pArpRequest->byProtocolAddressLength == EC_IPv4_ADDRESS_LEN)
            && (EC_NTOHS(pArpRequest->wOpCode) == EC_ARP_OPCODE_REQUEST))
        {
            /* only answer ARP request if IP address matches */
            if ((BroadcastEthernetAddress == pRequest->Destination)
                && (pArpRequest->Address.IpV4.DestinationIp == IpV4Address))
            {
                pArpReply->Address.IpV4.DestinationMac = pArpRequest->Address.IpV4.
↪SourceMac;
```

(continues on next page)

(continued from previous page)

```
                pArpReply->Address.IpV4.SourceMac = MacAddress;

                pArpReply->Address.IpV4.DestinationIp = pArpRequest->Address.IpV4.
↪SourceIp;
                pArpReply->Address.IpV4.SourceIp = IpV4Address;

                pArpReply->wOpCode = EC_HTONS(EC_ARP_OPCODE_REPLY);

                /* reply valid */
                pReplyFrameDesc->dwSize = ETHERNET_FRAME_LEN + EC_ARP_IPv4_HEADER_
↪LEN;
            }
        }
    } break;
    case ETHERNET_FRAME_TYPE_IP:
    {
        if (((EC_IP_HEADER*)EC_ENDOF(pRequest))->byProtocol == EC_IP_PROTOCOL_ICMP)
        {
            EC_ICMP_HEADER* pIcmpRequest = (EC_ICMP_HEADER*)EC_ENDOF(pRequest); /*␣
↪skip ETHERNET_FRAME header */
            EC_ICMP_HEADER* pIcmpReply = (EC_ICMP_HEADER*)EC_ENDOF(pReply); /*␣
↪skip ETHERNET_FRAME header */

            /* only answer PING request if MAC address and IP address match */
            if ((pIcmpRequest->byType == EC_ICMP_TYPE_ECHO)
                && (pRequest->Destination == MacAddress)
                && (pIcmpRequest->IpHdr.dwDstAddr == IpV4Address))
            {
                EC_T_WORD wLen = EC_NTOHS(pIcmpRequest->IpHdr.wTotalLength);

                /* swap src and dest ip address */
                pIcmpReply->IpHdr.dwSrcAddr = pIcmpRequest->IpHdr.dwDstAddr;
                pIcmpReply->IpHdr.dwDstAddr = pIcmpRequest->IpHdr.dwSrcAddr;

                pIcmpReply->byType = EC_ICMP_TYPE_ECHO_REPLY;

                pIcmpReply->IpHdr.wCheckSum = 0;
#if __GNUC__ >= 9
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Waddress-of-packed-member"
#endif
                pIcmpReply->IpHdr.wCheckSum = EC_HTONS(EcCalculateCrcRfc1071((EC_T_
↪WORD*)&pIcmpReply->IpHdr, EC_IP_HEADER_MINIMUM_LEN));
#if __GNUC__ >= 9
#pragma GCC diagnostic pop
#endif

                pIcmpRequest->wCheckSum = 0;
                pIcmpRequest->wCheckSum = EC_HTONS(EcCalculateCrcRfc1071((EC_T_
↪WORD*)&pIcmpReply->byType /* skip IpHdr */, (EC_T_WORD)(wLen - EC_IP_HEADER_
↪MINIMUM_LEN)));

                /* reply valid */
                pReplyFrameDesc->dwSize = (EC_T_WORD)(ETHERNET_FRAME_LEN + wLen);
            }
        }
    } break;
    }
}

static EC_T_VOID myAppEoe_APPL_EoeReceive(EC_T_VOID* pvContext, EC_T_DWORD␣
↪dwSimulatorId, EC_T_WORD wCfgFixedAddress, EC_T_WORD* pwFrame, EC_T_WORD␣
↪wFrameSize)
```

(continues on next page)

(continued from previous page)

```
{
    T_MY_EOE_CONTEXT* poContext = (T_MY_EOE_CONTEXT*)pvContext;
    EC_T_LINK_FRAMEDESC oReceiveFrame;
    oReceiveFrame.pbyFrame = (EC_T_BYTE*)pwFrame;
    oReceiveFrame.dwSize = wFrameSize;

    /* call ARP request and ping request handler and send reply if filled by␣
↪handler */
    myAppProcessEoeFrameArpAndPing(&oReceiveFrame, &poContext->oSendFrame,␣
↪poContext->oMacAddress, poContext->oIpV4Address);
    if (poContext->oSendFrame.dwSize > 0)
    {
        esEoeSendFrame(dwSimulatorId, wCfgFixedAddress, poContext->oSendFrame.
↪pbyFrame, poContext->oSendFrame.dwSize);
    }
}

static EC_T_VOID myAppEoe_APPL_EoeSettingInd(EC_T_VOID* pvContext, EC_T_DWORD /*␣
↪dwSimulatorId */, EC_T_WORD /* wCfgFixedAddress */, EC_T_WORD* pwMacAddress, EC_
↪T_WORD* pwIpV4Address,
    EC_T_WORD* /* pwIpV4SubNetMask */, EC_T_WORD* /* pwIpV4DefaultGateway */, EC_T_
↪WORD* /* pwDnsIpV4Address */)
{
    T_MY_EOE_CONTEXT* poContext = (T_MY_EOE_CONTEXT*)pvContext;
    poContext->oMacAddress = *((ETHERNET_ADDRESS*)pwMacAddress);
    poContext->oIpV4Address.dwAddr = EC_GETDWORD(pwIpV4Address);
}

static EC_INLINESTART EC_T_DWORD myAppEoeInit(T_EC_DEMO_APP_CONTEXT* pAppContext,␣
↪T_MY_EOE_CONTEXT* poContext, EC_T_WORD wSlaveAddress)
{
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_DWORD dwRes    = EC_E_ERROR;

    poContext->pAppContext = pAppContext;
    poContext->wSlaveAddress = wSlaveAddress;

    poContext->oSendFrame.pbyFrame = (EC_T_BYTE*)OsMalloc(ETHERNET_MAX_FRAMEBUF_
↪LEN /* 1536 */);
    if (EC_NULL == poContext->oSendFrame.pbyFrame)
    {
        dwRetVal = EC_E_NOMEMORY;
        goto Exit;
    }

    {
        struct _EC_T_SLAVE_SSC_APPL_DESC oSlaveApplDesc;
        OsMemset(&oSlaveApplDesc, 0, sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC));

        oSlaveApplDesc.dwSignature = SIMULATOR_SIGNATURE;
        oSlaveApplDesc.dwSize = sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC);
        oSlaveApplDesc.szName = (EC_T_CHAR*)"MyAppEoeSlaveAppl";
        oSlaveApplDesc.pvContext = poContext; /* first parameter of static wrapper␣
↪functions */

        oSlaveApplDesc.pfnEoeReceive = myAppEoe_APPL_EoeReceive;
        oSlaveApplDesc.pfnEoeSettingInd = myAppEoe_APPL_EoeSettingInd;

        /* register callbacks at slave */
        dwRes = esSetSlaveSscApplication(pAppContext->dwInstanceId, wSlaveAddress,␣
↪&oSlaveApplDesc);
```

(continues on next page)

(continued from previous page)

```
        if (EC_E_NOERROR != dwRes)
        {
            dwRetVal = dwRes;
            goto Exit;
        }
    }

    dwRetVal = EC_E_NOERROR;
Exit:
    if (EC_E_NOERROR != dwRetVal)
    {
        OsSafeFree(poContext->oSendFrame.pbyFrame);
    }

    return dwRetVal;
} EC_INLINESTOP
```

The callbacks need a context. Each SubDevice needs its individual context. If there is only one SubDevice to be registered, a global context can be declared and used:

```
T_MY_EOE_CONTEXT G_oContext;
```

The callbacks must be registered using *esSetSlaveSscApplication()*, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo:

```
    OsMemset(&G_oContext, 0, sizeof(G_oContext));
    dwRes = myAppEoeInit(pAppContext, &G_oContext, wSlaveAddress);
    if (EC_E_NOERROR != dwRes)
    {
        dwRetVal = dwRes;
        goto Exit;
    }
```

6.13 File access over EtherCAT® (FoE)

The following examples demonstrate how to customize FoE simulation using *esSetSlaveSscApplication()*.

FoE Download Example

The following code demonstrates how to receive FoE from the MainDevice and store it in a buffer. The data received in this example is stored in T_MY_CONTEXT::abyFileBuf :

```
#define MYAPP_FOE_BUFFER_SIZE (35264)
typedef struct
{
    EC_T_BYTE abyFileBuf[MYAPP_FOE_BUFFER_SIZE];
    EC_T_WORD wFileBufLen;

    /* ... */
} T_MY_CONTEXT;
```

The callback myAppFoeWrite() handles download requests from the MainDevice, myAppFoeWriteData() copies the FoE payload from the mailbox to T_MY_CONTEXT::abyFileBuf :

```
extern "C" EC_T_WORD EC_FNCALL myAppFoeWrite(
    EC_T_VOID* pvContext, EC_T_DWORD /* dwSimulatorId */, EC_T_WORD /*␣
↪wCfgFixedAddress */,
    EC_T_WORD*, EC_T_WORD, EC_T_DWORD)
{
    T_MY_CONTEXT* poContext = (T_MY_CONTEXT*)pvContext;
    poContext->wFileBufLen = 0;

    return 0; /* accept file download (ECAT_FOE_ERRCODE_... denies download) */
}

extern "C" EC_T_WORD EC_FNCALL myAppFoeWriteData(
    EC_T_VOID* pvContext, EC_T_DWORD /* dwSimulatorId */, EC_T_WORD /*␣
↪wCfgFixedAddress */,
    EC_T_WORD* pData, EC_T_WORD wSize, EC_T_BYTE bDataFollowing)
{
    T_MY_CONTEXT* poContext = (T_MY_CONTEXT*)pvContext;
    if (poContext->wFileBufLen + wSize > sizeof(poContext->abyFileBuf))
    {
        return ECAT_FOE_ERRCODE_DISKFULL; /* abort transfer */
    }

    OsMemcpy(&poContext->abyFileBuf[poContext->wFileBufLen], pData, wSize);
    poContext->wFileBufLen = poContext->wFileBufLen + wSize;

    /* FoE ACK segment at Master (more segments follow / download finished) */
    return bDataFollowing ? SSC_FOE_ACK : SSC_FOE_ACKFINISHED;
}
```

See type EC_PF_SSC_APPL_FOE_WRITE .

The callbacks must be registered using *esSetSlaveSscApplication()*, e.g. in myAppPrepare() of EcSimulatorHilDemo / EcSimulatorSilDemo:

```
struct _EC_T_SLAVE_SSC_APPL_DESC oSlaveApp;
OsMemset(&oSlaveApp, 0, sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC));

T_MY_CONTEXT* poSlaveApplContext = &G_oSlaveApplContext;
OsMemset(poSlaveApplContext, 0, sizeof(G_oSlaveApplContext));
```

(continues on next page)

(continued from previous page)

```
oSlaveApp.dwSignature = SIMULATOR_SIGNATURE;
oSlaveApp.dwSize = sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC);
oSlaveApp.szName = (EC_T_CHAR*)"mySlaveAppl";
oSlaveApp.pvContext = poSlaveApplContext; /* pvContext of callbacks */
oSlaveApp.pfnFoeWrite = myAppFoeWrite;
oSlaveApp.pfnFoeWriteData = myAppFoeWriteData;

/* register SlaveSscApplication call-backs (Master in INIT) */
dwRes = esSetSlaveSscApplication(dwSimulatorId, wSlaveAddress, &oSlaveApp);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
```

The following code at MainDevice downloads the file to the SubDevice:

```
/* set Master PREOP */
dwRes = emSetMasterState(dwMasterId, 30000 /* dwTimeout */, eEcatState_PREOP);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}

/* download file */
dwRes = emFoeFileDownload(dwMasterId, emGetSlaveId(dwMasterId, wSlaveAddress),
    (EC_T_CHAR*)"foe.dat", (EC_T_DWORD)OsStrlen((EC_T_CHAR*)"foe.dat"),
    G_abyFileBuf, MYAPP_FOE_BUFFER_SIZE, 0 /* dwPassword */, 30000 /* dwTimeout */
↪);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
```

FoE Upload Example

The following code demonstrates how to send FoE data from a local buffer to the MainDevice. The data sent in this example is stored in `T_MY_CONTEXT::abyFileBuf`:

```
#define MYAPP_FOE_BUFFER_SIZE (35264)
typedef struct
{
    EC_T_BYTE abyFileBuf[MYAPP_FOE_BUFFER_SIZE];
    EC_T_WORD wFileBufLen;

    /* ... */
} T_MY_CONTEXT;
```

`myAppFoeRead()` handles upload requests from the MainDevice, `myAppFoeReadData()` copies the FoE payload from `T_MY_CONTEXT::abyFileBuf` to the mailbox:

```
extern "C" EC_T_WORD EC_FNCALL myAppFoeReadData(
    EC_T_VOID* pvContext, EC_T_DWORD /* dwSimulatorId */, EC_T_WORD /*␣
↪wCfgFixedAddress */,
    EC_T_DWORD dwOffset, EC_T_WORD wMaxBlockSize, EC_T_WORD* pData)
{
    T_MY_CONTEXT* poContext = (T_MY_CONTEXT*)pvContext;
    if (dwOffset > poContext->wFileBufLen)
    {
        return 0;
    }

    if (dwOffset + wMaxBlockSize > poContext->wFileBufLen)
    {
        wMaxBlockSize = EC_LOWORD(poContext->wFileBufLen - dwOffset);
    }

    if (wMaxBlockSize > 0)
    {
        OsMemcpy(pData, &poContext->abyFileBuf[dwOffset], wMaxBlockSize);
    }

    return wMaxBlockSize; /* amount of data read */
}

extern "C" EC_T_WORD EC_FNCALL myAppFoeRead(
    EC_T_VOID* pvContext, EC_T_DWORD dwSimulatorId, EC_T_WORD wCfgFixedAddress,
    EC_T_WORD*, EC_T_WORD, EC_T_DWORD, EC_T_WORD wMaxBlockSize, EC_T_WORD* pData)
{
    /* APPL_FoeRead contains returning first data */
    return myAppFoeReadData(pvContext, dwSimulatorId, wCfgFixedAddress, 0,␣
↪wMaxBlockSize, pData);
}
```

The callbacks must be registered using *esSetSlaveSscApplication()*, e.g. in `myAppPrepare()` of EcSimulatorHilDemo / EcSimulatorSilDemo:

```
    struct _EC_T_SLAVE_SSC_APPL_DESC oSlaveApp;
    OsMemset(&oSlaveApp, 0, sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC));

    T_MY_CONTEXT* poSlaveApplContext = &G_oSlaveApplContext;
    OsMemset(poSlaveApplContext, 0, sizeof(G_oSlaveApplContext));

    /* pattern for file content */
    OsMemset(poSlaveApplContext->abyFileBuf, 0xAA, MYAPP_FOE_BUFFER_SIZE);
    poSlaveApplContext->wFileBufLen = MYAPP_FOE_BUFFER_SIZE;
```

(continues on next page)

(continued from previous page)

```
    oSlaveApp.dwSignature = SIMULATOR_SIGNATURE;
    oSlaveApp.dwSize = sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC);
    oSlaveApp.szName = (EC_T_CHAR*)"mySlaveAppl";
    oSlaveApp.pvContext = poSlaveApplContext; /* pvContext of callbacks */
    oSlaveApp.pfnFoeRead = myAppFoeRead;
    oSlaveApp.pfnFoeReadData = myAppFoeReadData;

    /* register SlaveSscApplication callbacks (Master in INIT) */
    dwRes = esSetSlaveSscApplication(dwSimulatorId, wSlaveAddress, &oSlaveApp);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
```

The following code at MainDevice uploads the file from the SubDevice:

```
    /* set Master PREOP */
    dwRes = emSetMasterState(dwMasterId, 30000 /* dwTimeout */, eEcatState_PREOP);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }

    /* upload file */
    dwRes = emFoeFileUpload(dwMasterId, emGetSlaveId(dwMasterId, wSlaveAddress),
        (EC_T_CHAR*)"foe.dat", (EC_T_DWORD)OsStrlen((EC_T_CHAR*)"foe.dat"),
        G_abyFileBuf, MYAPP_FOE_BUFFER_SIZE, &dwOutDataLen, 0 /* dwPassword */,␣
↪30000 /* dwTimeout */);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
```

See also:

- *esInitSimulator()*
- *esConfigureNetwork()*

6.14 Vendor specific access over EtherCAT® (VoE)

See also *VoE Receive Example*

6.14.1 esVoeSend

EC_T_DWORD **esVoeSend** (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    EC_T_WORD wDstFixedAddress,
    const EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen
)

Fill “Mailbox In” (SM1) with VoE data to be polled by Master. The slave must have VoE enabled, see ESI: /EtherCATInfo/Descriptions/Devices/Device/Mailbox/VoE, ENI/EXI: /EtherCATConfig/Config/Slave/Mailbox/Protocol/VoE.

–

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave’s station address
- **wDstFixedAddress** – [in] Destination address
- **pbyData** – [in] Data
- **dwDataLen** – [in] Data length

Returns
EC_E_NOERROR or error code

6.14.2 esSetVoeReceiveCallback

EC_T_DWORD **esSetVoeReceiveCallback** (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    EC_T_PF_VOE_RECEIVE_CB pfVoeReceive,
    EC_T_VOID *pvContext
)

Set VoE data received callback for “Mailbox Out” (SM0). Within the callback, no EC-Simulator API may be called. The slave must have VoE enabled, see ESI: /EtherCATInfo/Descriptions/Devices/Device/Mailbox/VoE, ENI/EXI: /EtherCATConfig/Config/Slave/Mailbox/Protocol/VoE.

Parameters

- **dwInstanceId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave’s station address
- **pfVoeReceive** – [in] Receive callback function
- **pvContext** – [in] Receive callback function context

Returns

- *EC_E_NOERROR*: on success
- *EC_E_NOTFOUND*: if slave not found in ENI/EXI (wCfgFixedAddress)
- *EC_E_NO_MBX_SUPPORT*: if slave without Mailbox support

6.14.3 VoE Receive Example

The following examples demonstrate how to register a handler for VoE writes from the MainDevice or from other SubDevices.

VoE Receive Example

The following code demonstrates how to send VoE from the MainDevice to the SubDevice and store it in a buffer at the Simulator. The data received in this example is stored in `T_MY_CONTEXT::abyVoeBuf`:

```
#define MYAPP_VOE_BUFFER_SIZE (512)
typedef struct
{
    EC_T_BYTE  abyVoeBuf[MYAPP_VOE_BUFFER_SIZE];
    EC_T_DWORD dwVoeBufLen;
    EC_T_VOID* pvEvent;

    /* ... */
} T_MY_CONTEXT;
```

`myAppVoeReceiveCallback()` handles write requests from the MainDevice and copies the received VoE payload from the mailbox to `T_MY_CONTEXT::abyVoeBuf`:

```
EC_T_BYTE EC_FNCALL myAppVoeReceiveCallback(
    EC_T_VOID* pvContext, EC_T_DWORD /* dwSimulatorId */,
    EC_T_WORD /* wCfgFixedAddress VoE receiver address, e.g. 1001 */,
    EC_T_WORD /* wSrcFixedAddress VoE sender address, 0: Master */,
    const EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen)
{
    T_MY_CONTEXT* poContext = (T_MY_CONTEXT*)pvContext;

    OsMemcpy(poContext->abyVoeBuf, pbyData, EC_AT_MOST(MYAPP_VOE_BUFFER_SIZE,␣
↪dwDataLen));
    poContext->dwVoeBufLen = dwDataLen;

    /* No EC-Simulator APIs may be issued from within the callback context.
       Signalize VoE data received for deferred processing in other thread. */
    OsSetEvent(poContext->pvEvent);

    return 0;
}
```

The following code shows how to register `myAppVoeReceiveCallback()` and send VoE from the MainDevice to the simulated SubDevice:

```
    EC_T_DWORD dwRes = EC_E_ERROR;
    EC_T_DWORD dwRetVal = EC_E_ERROR;

#define MYAPP_VOE_SEND_BUF_LEN sizeof(EC_T_DWORD)
    EC_T_BYTE abyVoeSendBuf[MYAPP_VOE_SEND_BUF_LEN];
    OsMemset(abyVoeSendBuf, 0, MYAPP_VOE_SEND_BUF_LEN);

    T_MY_CONTEXT oContext;
    OsMemset(&oContext, 0, sizeof(T_MY_CONTEXT));
```

(continues on next page)

(continued from previous page)

```
    oContext.pvEvent = OsCreateEvent();
    if (EC_NULL == oContext.pvEvent)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"OsCreateEvent failed\n"));
        dwRetVal = EC_E_NOMEMORY;
        goto Exit;
    }

    /* EC-Simulator: register VoE handler */
    dwRes = esSetVoeReceiveCallback(dwSimulatorId, 1001, myAppVoeReceiveCallback, &
↪oContext);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"esSetVoeReceiveCallback failed: %s (0x%lx))\n", esGetText(dwSimulatorId, dwRes),
↪ dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* EC-Master: set PREOP state */
    dwRes = emSetMasterState(dwMasterId, 30000, eEcatState_PREOP);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emSetMasterState failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* send VoE to slave */
    EC_SET_FRM_DWORD(abyVoeSendBuf, 0x12345678);
    dwRes = emVoeWrite(dwMasterId, emGetSlaveId(dwMasterId, 1001), abyVoeSendBuf,␣
↪MYAPP_VOE_SEND_BUF_LEN, 5000);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emVoeWrite failed: %s (0x%lx))\n", ecatGetText(dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    /* wait for slave received data in callback */
    dwRes = OsWaitForEvent(oContext.pvEvent, 5000);
    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"OsWaitForEvent failed: %s (0x%lx))\n", esGetText(dwSimulatorId, dwRes), dwRes));
        dwRetVal = dwRes;
        goto Exit;
    }

    dwRetVal = EC_E_NOERROR;
Exit:
    OsSafeDeleteEvent(oContext.pvEvent);
```

6.15 Distributed Clocks (DC)

Distributed Clocks are rudimentarily supported.

7 Error Codes

7.1 Groups

No.	Group	Abbr.	Description
1	Application Error	APP	Error within application, running the Simulator E.g. API function call with invalid parameters
2	EtherCAT® network information file problem	ENI	Network configuration XML file error E.g. malformed or empty EtherCAT® Network configuration file
3	Simulator parameter configuration	CFG	Simulator configuration parameters erroneous E.g. mailbox command queue not large enough
4	Bus/SubDevice Error	SLV	SubDevice error E.g. Working Counter Error
5	Real-time Ethernet Driver	LLA	Real-time Ethernet Driver error (network interface driver) E.g. Intel Pro/1000 NIC could not be found
6	Remote API	RAS	Remote API error E.g. connection to Remote API server is not possible from client
7	Internal software error	ISW	Simulator internal error

7.2 Generic Error Codes

EC_E_NOERROR
0x00000000: No Error

EC_E_ERROR
0x98110000: Unspecific Error

EMRAS_E_ERROR
0x98110180: Unspecific RAS Error

EC_E_NOTSUPPORTED
0x98110001: APP: Feature not supported (e.g. function or property not available)

EC_E_INVALIDINDEX
0x98110002: APP: Invalid index (e.g. CoE: invalid SDO index)

EC_E_INVALIDOFFSET
0x98110003: ISW: Invalid offset (e.g. invalid offset while accessing Process Data Image)

EC_E_CANCEL
0x98110004: APP: Cancel (e.g. EtherCAT stack should abort current mailbox transfer)

EC_E_INVALIDSIZE
0x98110005: APP: Invalid size

EC_E_INVALIDDATA
0x98110006: ISW: Invalid data (multiple error sources)

EC_E_NOTREADY
0x98110007: ISW: Not ready (multiple error sources)

EC_E_BUSY
0x98110008: APP: Busy (e.g. EtherCAT stack is currently busy and not available to process the API request. The function may be called again later)

EC_E_ACYC_FRM_FREEQ_EMPTY
0x98110009: ISW: Cannot queue acyclic EtherCAT command (Acyclic command queue is full. Possible solution: Increase of configuration value dwMaxQueuedEthFrames)

EC_E_NOMEMORY
0x9811000A: CFG: No memory left (e.g. memory full / fragmented))

EC_E_INVALIDPARM
0x9811000B: APP: Invalid parameter (e.g. API function called with erroneous parameter set)

EC_E_NOTFOUND
0x9811000C: APP: Not found (e.g. Network Information File ENI not found or API called with invalid slave ID)

EC_E_DUPLICATE
0x9811000D: ISW: Duplicated fixed address detected (handled internally)

EC_E_INVALIDSTATE
0x9811000E: ISW: Invalid state (EtherCAT stack not initialized or not configured)

EC_E_TIMER_LIST_FULL
0x9811000F: ISW: Cannot add slave to timer list (slave timer list full)

EC_E_TIMEOUT
0x98110010: Timeout

EC_E_OPENFAILED
0x98110011: ISW: Open failed

EC_E_SENDFAILED
0x98110012: LLA: Frame send failed

EC_E_INSERTMAILBOX
0x98110013: CFG: Insert Mailbox error (internal limit MAX_QUEUED_COE_CMDS: 20)

EC_E_INVALIDCMD
0x98110014: ISW: Invalid Command (Unknown mailbox command code)

EC_E_UNKNOWN_MBX_PROTOCOL
0x98110015: ISW: Unknown Mailbox Protocol Command (Unknown Mailbox protocol or mailbox command with unknown protocol association)

EC_E_ACCESSDENIED
0x98110016: ISW: Access Denied (e.g. master internal software error)

EC_E_IDENTIFICATIONFAILED
0x98110017: ENI: Identification failed (e.g. identification command failed)

EC_E_LOCK_CREATE_FAILED
0x98110018: SYS: Create lock failed (e.g. OsCreateLockTyped failed)

EC_E_VERSION_MISMATCH
0x98110019: APP: Version mismatch for loaded library

EC_E_PRODKEY_INVALID
0x9811001A: CFG: Product Key Invalid (e.g. application using protected version of the stack, which stops operation after the evaluation time limit reached if a license is not provided)

EC_E_WRONG_FORMAT
0x9811001B: ENI: Wrong configuration format (e.g. Network information file empty or malformed), SLV: Malformed EEPROM content

EC_E_FEATURE_DISABLED
0x9811001C: APP: Feature disabled (e.g. Application tried to perform a missing or disabled API function)

EC_E_SHADOW_MEMORY
0x9811001D: Shadow memory requested in wrong mode

EC_E_BUSCONFIG_MISMATCH
0x9811001E: ENI: Bus configuration mismatch (e.g. Network information file and currently connected bus topology does not match)

EC_E_CONFIGDATAREAD
0x9811001F: ENI: Error reading configuration file (e.g. Network information file could not be read)

EC_E_ENI_NO_SAFEOP_OP_SUPPORT
0x98110020: Configuration doesn't support SAFEOP and OP requested state

EC_E_XML_CYCCMDS_MISSING
0x98110021: ENI: Cyclic commands are missing (e.g. Network information file does not contain cyclic commands)

EC_E_XML_ALSTATUS_READ_MISSING
0x98110022: ENI: AL_STATUS register read missing in XML file for at least one state (e.g. Read of AL Status register is missing in cyclic part of given network information file)

EC_E_MCSM_FATAL_ERROR
0x98110023: ISW: Fatal internal McSm (master control state machine is in an undefined state)

EC_E_SLAVE_ERROR
0x98110024: SLV: Slave error (e.g. A slave error was detected. See also EC_NOTIFY_STATUS_SLAVE_ERROR and EC_NOTIFY_SLAVE_ERROR_STATUS_INFO)

EC_E_FRAME_LOST
0x98110025: SLV: Frame lost, IDX mismatch (EtherCAT frame(s) lost on bus, means the response was not received. In case this error shows frequently a problem with the wiring could be the cause)

EC_E_CMD_MISSING
0x98110026: SLV: At least one EtherCAT command is missing in the received frame (e.g. received EtherCAT frame incomplete)

EC_E_CYCCMD_WKC_ERROR
0x98110027: Cyclic command WKC error

EC_E_INVALID_DCL_MODE
0x98110028: APP: IOCTL EC_IOCTL_DC_LATCH_REQ_LTIMVALS invalid in DCL auto read mode (this function cannot be used if DC Latching is running in mode “Auto Read”)

EC_E_AI_ADDRESS
0x98110029: SLV: Auto increment address increment mismatch (e.g. Network information file and bus topology doesn’t match any more. Error shows only, if an already recognized slave isn’t present any more)

EC_E_INVALID_SLAVE_STATE
0x9811002A: APP: Slave in invalid state, e.g. not in OP (API not callable in this state) (mailbox commands are not allowed in current slave state)

EC_E_SLAVE_NOT_ADDRESSABLE
0x9811002B: SLV: Station address lost (or slave missing) - FPRD to AL_STATUS failed (e.g. Slave had a power cycle)

EC_E_CYC_CMDS_OVERFLOW
0x9811002C: ENI: Too many cyclic commands in XML configuration file (e.g. EC_T_INIT_MASTER_PARMS.dwMaxAcycFramesQueued too small)

EC_E_LINK_DISCONNECTED
0x9811002D: SLV: Ethernet link cable disconnected (e.g. EtherCAT bus segment not connected to network interface)

EC_E_MASTERCORE_INACCESSIBLE
0x9811002E: RAS: Master core not accessible (e.g. Connection to remote server was terminated or master instance has been stopped on remote side)

EC_E_COE_MBXSND_WKC_ERROR
0x9811002F: SLV: CoE mailbox send: working counter (e.g. CoE mailbox couldn’t be read on slave, slave didn’t read out mailbox since last write)

EC_E_COE_MBXRCV_WKC_ERROR
0x98110030: SLV: CoE mailbox receive: working counter (e.g. CoE mailbox couldn’t be read from slave)

EC_E_NO_MBX_SUPPORT
0x98110031: APP: No mailbox support (e.g. Slave does not support mailbox access)

EC_E_NO_COE_SUPPORT
0x98110032: ENI: CoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_EOE_SUPPORT
0x98110033: ENI: EoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_FOE_SUPPORT
0x98110034: ENI: FoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_SOE_SUPPORT
0x98110035: ENI: SoE protocol not supported (e.g. Configuration error or slave information file doesn't match slave firmware)

EC_E_NO_VOE_SUPPORT
0x98110036: ENI: VoE protocol not supported (e.g. Configuration error or slave information file doesn't match slave firmware)

EC_E_EVAL_VIOLATION
0x98110037: ENI: Configuration violates Evaluation limits (obsolete)

EC_E_EVAL_EXPIRED
0x98110038: CFG: Evaluation Time limit reached (e.g. License not provided and evaluation period (1 hour) of protected version exceeded)

EC_E_LICENSE_MISSING
0x98110039: License key invalid or missing

EC_E_CFGFILENOTFOUND
0x98110070: CFG: Network configuration file not found (e.g. path to configuration file (XML) was wrong or the file is not available)

EC_E_EEPROMREADERROR
0x98110071: SLV: Command error while EEPROM upload (read slave EEPROM)

EC_E_EEPROMWRITEERROR
0x98110072: SLV: Command error while EEPROM download (write slave EEPROM)

EC_E_XML_CYCCMDS_SIZEMISMATCH
0x98110073: ENI: Cyclic command wrong size (too long) (size in network configuration file (XML) does not match size of process data)

EC_E_XML_INVALID_INP_OFF
0x98110074: ENI: Invalid input offset in cyclic command, please check InputOffs

EC_E_XML_INVALID_OUT_OFF
0x98110075: ENI: Invalid output offset in cyclic command, please check OutputOffs

EC_E_PORTCLOSE
0x98110076: Port close failed

EC_E_PORTOPEN
0x98110077: Port open failed

EC_E_SLAVE_NOT_PRESENT
0x9811010E: APP / SLV: command not executed (slave not present on bus) (e.g. slave disappeared or was never present)

EC_E_EEPROMRELOADERROR
0x98110110: Command error while EEPROM reload

EC_E_SLAVECTRLRESETERROR
0x98110111: Command error while Reset Slave Controller

EC_E_SYSDRIVERMISSING
0x98110112: SYS: Cannot open system driver (e.g. system driver was not loaded)

EC_E_BUSCONFIG_TOPOCHANGE
0x9811011E: Bus configuration not detected, Topology changed (e.g. Topology changed while scanning bus)

EC_E_EOE_MBX_WKC_ERROR
0x9811011F: EoE: Mailbox receive: working counter

EC_E_FOE_MBX_WKC_ERROR
0x98110120: FoE: Mailbox receive: working counter

EC_E_SOE_MBX_WKC_ERROR
0x98110121: SoE: mailbox receive: working counter

EC_E_AOE_MBX_WKC_ERROR
0x98110122: AoE: Mailbox receive: working counter

EC_E_VOE_MBX_WKC_ERROR
0x98110123: SLV: VoE mailbox send: working counter (VoE mailbox couldn't be written)

EC_E_EEPROMASSIGNERROR
0x98110124: SLV: EEPROM assignment failed

EC_E_MBX_ERROR_TYPE
0x98110125: SLV: Unknown mailbox error code received in mailbox

EC_E_REDLINEBREAK
0x98110126: SLV: Redundancy line break (e.g. cable break between slaves or between EtherCAT network adapter and first slave)

EC_E_XML_INVALID_CMD_WITH_RED
0x98110127: ENI: Invalid EtherCAT command in cyclic frame with redundancy (e.g. BRW commands are not allowed with redundancy)

EC_E_XML_PREV_PORT_MISSING
0x98110128: ENI: <PreviousPort>-tag is missing (e.g. if the auto increment address is not the first slave on the bus we check if a previous port tag OR a hot connect tag is available)

EC_E_XML_DC_CYCCMDS_MISSING
0x98110129: DC enabled and DC cyclic commands missing (e.g. access to 0x0900)

EC_E_DLSTATUS_IRQ_TOPOCHANGED
0x98110130: SLV: Data link (DL) status interrupt because of changed topology (automatically handled by master)

EC_E_PTS_IS_NOT_RUNNING
0x98110131: PTS: Pass Through Server is not running (Pass-Through-Server was tried to be enabled/disabled or stopped without being started)

EC_E_PTS_IS_RUNNING
0x98110132: PTS: Pass Through Server is running (obsolete, replaced by EC_E_ADS_IS_RUNNING)

EC_E_ADS_IS_RUNNING
0x98110132: PTS: ADS adapter (Pass Through Server) is running (API call conflicts with ADS state (running))

EC_E_PTS_THREAD_CREATE_FAILED
0x98110133: PTS: Could not start the Pass Through Server

EC_E_PTS_SOCK_BIND_FAILED
0x98110134: PTS: The Pass Through Server could not bind the IP address with a socket (e.g. Possibly because the IPaddress (and Port) is already in use or the IP-address does not exist)

EC_E_PTS_NOT_ENABLED
0x98110135: PTS: The Pass Through Server is running but not enabled

EC_E_PTS_LL_MODE_NOT_SUPPORTED
0x98110136: PTS: The Link Layer mode is not supported by the Pass Through Server (e.g. The Master is running in interrupt mode but the Pass-Through-Server only supports polling mode)

EC_E_VOE_NO_MBX_RECEIVED
0x98110137: SLV: No VoE mailbox received yet from specific slave

EC_E_DC_REF_CLOCK_SYNC_OUT_UNIT_DISABLED
0x98110138: DC (time loop control) unit of reference clock disabled

EC_E_DC_REF_CLOCK_NOT_FOUND
0x98110139: SLV: Reference clock not found! May happen if reference clock is removed from network.

EC_E_MBX_CMD_WKC_ERROR
0x9811013B: SLV: Mailbox command working counter error (e.g. Mailbox init command Retry Count exceeded)

EC_E_NO_AOE_SUPPORT
0x9811013C: APP / SLV: AoE: Protocol not supported (e.g. Application calls AoE-API although not implemented at slave)

EC_E_AOE_INV_RESPONSE_SIZE
0x9811013D: AoE: Invalid AoE response received

EC_E_AOE_ERROR
0x9811013E: AoE: Common AoE device error

EC_E_AOE_SRVNOTSUPP
0x9811013F: AoE: Service not supported by server

EC_E_AOE_INVALIDGRP
0x98110140: AoE: Invalid index group

EC_E_AOE_INVALIDOFFSET
0x98110141: AoE: Invalid index offset

EC_E_AOE_INVALIDACCESS
0x98110142: AoE: Reading/writing not permitted

EC_E_AOE_INVALIDSIZE
0x98110143: AoE: Parameter size not correct

EC_E_AOE_INVALIDDATA
0x98110144: AoE: Invalid parameter value(s)

EC_E_AOE_NOTREADY
0x98110145: AoE: Device not in a ready state

EC_E_AOE_BUSY
0x98110146: AoE: Device busy

EC_E_AOE_INVALIDCONTEXT
0x98110147: AoE: Invalid context

EC_E_AOE_NOMEMORY
0x98110148: AoE: Out of memory

EC_E_AOE_INVALIDPARM
0x98110149: AoE: Invalid parameter value(s)

EC_E_AOE_NOTFOUND
0x9811014A: AoE: Not found

EC_E_AOE_SYNTAX
0x9811014B: AoE: Syntax error in command or file

EC_E_AOE_INCOMPATIBLE
0x9811014C: AoE: Objects do not match

EC_E_AOE_EXISTS
0x9811014D: AoE: Object already exists

EC_E_AOE_SYMBOLNOTFOUND
0x9811014E: AoE: Symbol not found

EC_E_AOE_SYMBOLVERSIONINVALID
0x9811014F: AoE: Symbol version invalid

EC_E_AOE_INVALIDSTATE
0x98110150: AoE: Server in invalid state

EC_E_AOE_TRANSMODENOTSUPP
0x98110151: AoE: AdsTransMode not supported

EC_E_AOE_NOTIFYHNDINVALID
0x98110152: AoE: Notification handle invalid

EC_E_AOE_CLIENTUNKNOWN
0x98110153: AoE: Notification client not registered

EC_E_AOE_NOMOREHDLS
0x98110154: AoE: No more notification handles

EC_E_AOE_INVALIDWATCHSIZE
0x98110155: AoE: Size for watch to big

EC_E_AOE_NOTINIT
0x98110156: AoE: Device not initialized

EC_E_AOE_TIMEOUT
0x98110157: AoE: Device has a timeout

EC_E_AOE_NOINTERFACE
0x98110158: AoE: Query interface failed

EC_E_AOE_INVALIDINTERFACE
0x98110159: AoE: Wrong interface required

EC_E_AOE_INVALIDCLSID
0x9811015A: AoE: Class ID invalid

EC_E_AOE_INVALIDOBJID
0x9811015B: AoE: Object ID invalid

EC_E_AOE_PENDING
0x9811015C: AoE: Request pending

EC_E_AOE_ABORTED
0x9811015D: AoE: Request aborted

EC_E_AOE_WARNING
0x9811015E: AoE: Signal warning

EC_E_AOE_INVALIDARRAYIDX
0x9811015F: AoE: Invalid array index

EC_E_AOE_SYMBOLNOTACTIVE
0x98110160: AoE: Symbol not active -> release handle and try again

EC_E_AOE_ACCESSDENIED
0x98110161: AoE: Access denied

EC_E_AOE_INTERNAL
0x98110162: AoE: Internal error

EC_E_AOE_TARGET_PORT_NOT_FOUND
0x98110163: AoE: Target port not found

EC_E_AOE_TARGET_MACHINE_NOT_FOUND
0x98110164: AoE: Target machine not found

EC_E_AOE_UNKNOWN_CMD_ID
0x98110165: AoE: Unknown command ID

EC_E_AOE_PORT_NOT_CONNECTED
0x98110166: AoE: Port not connected

EC_E_AOE_INVALID_AMS_LENGTH
0x98110167: AoE: Invalid AMS length

EC_E_AOE_INVALID_AMS_ID
0x98110168: AoE: invalid AMS Net ID

EC_E_AOE_PORT_DISABLED
0x98110169: AoE: Port disabled

EC_E_AOE_PORT_CONNECTED
0x9811016A: AoE: Port already connected

EC_E_AOE_INVALID_AMS_PORT
0x9811016B: AoE: Invalid AMS port

EC_E_AOE_NO_MEMORY
0x9811016C: AoE: No memory

EC_E_AOE_VENDOR_SPECIFIC
0x9811016D: AoE: Vendor specific AoE device error

EC_E_XML_AOE_NETID_INVALID
0x9811016E: ENI: AoE: Invalid NetID (e.g. Error from Configuration Tool)

EC_E_MAX_BUS_SLAVES_EXCEEDED
0x9811016F: CFG: Error: Maximum number of bus slave has been exceeded (The maximum number of preallocated bus slave objects is too small. The maximum number can be adjusted by the master initialization parameter EC_T_INITMASTERPARMS.dwMaxBusSlaves)

EC_E_MBXERR_SYNTAX
0x98110170: SLV: Mailbox error: Syntax of 6 octet Mailbox header is wrong (Slave error mailbox return value: 0x01)

EC_E_MBXERR_UNSUPPORTEDPROTOCOL
0x98110171: SLV: Mailbox error: The Mailbox protocol is not supported (Slave error mailbox return value: 0x02)

EC_E_MBXERR_INVALIDCHANNEL
0x98110172: SLV: Mailbox error: Field contains wrong value (Slave error mailbox return value: 0x03)

EC_E_MBXERR_SERVICENOTSUPPORTED
0x98110173: SLV: Mailbox error: The addressed service in the mailbox protocol is not supported (Slave error mailbox return value: 0x04)

EC_E_MBXERR_INVALIDHEADER
0x98110174: SLV: Mailbox error: The mailbox protocol header of the mailbox protocol is wrong (Slave error mailbox return value: 0x05)

EC_E_MBXERR_SIZETOOSHORT
0x98110175: SLV: Mailbox error: Length of received mailbox data is too short (Slave error mailbox return value: 0x06)

EC_E_MBXERR_NOMOREMEMORY
0x98110176: SLV: Mailbox error: Mailbox protocol can not be processed because of limited resources (Slave error mailbox return value: 0x07)

EC_E_MBXERR_INVALIDSIZE
0x98110177: SLV: Mailbox error: The length of data is inconsistent (Slave error mailbox return value: 0x08)

EC_E_DC_SLAVES_BEFORE_REF_CLOCK
0x98110178: ENI: Slaves with DC configured present on bus before reference clock (e.g. The first DC Slave was not configured as potential reference clock)

EC_E_DATA_TYPE_CONVERSION_FAILED
0x98110179: Data type conversion failed

EC_E_LINE_CROSSED
0x9811017B: Line crossed (cabling wrong)

EC_E_LINE_CROSSED_SLAVE_INFO
0x9811017C: Line crossed at slave (obsolete)

EC_E_ADO_NOT_SUPPORTED
0x9811017E: SLV: ADO for slave identification not supported (e.g. Request ID mechanism (ADO 0x134) not supported by slave)

EC_E_FRAMELOSS_AFTER_SLAVE
0x9811017F: Frameloss after Slave (opening port destroys communication)

EC_E_OEM_SIGNATURE_MISMATCH
0x98130008: ENI, OEM: Manufacturer signature mismatch

EC_E_ENI_ENCRYPTION_WRONG_VERSION
0x98130009: ENI, OEM: ENI encryption algorithm version not supported

EC_E_ENI_ENCRYPTED
0x9813000A: OEM: Loading encrypted ENI needs OEM key

EC_E_OEM_KEY_MISMATCH
0x9813000B: RAS, APP: OEM key mismatch

EC_E_OEM_KEY_MISSING
0x9813000C: APP: OEM key access needs OEM key set (e.g. Application must call esSetOemKey (HiL) or set *EC_T_LINK_PARMS_SIMULATOR::qwOemKey* (SiL))

EC_E_S2SMBX_NOT_CONFIGURED
0x98130020: S2S: Not Configured

EC_E_S2SMBX_NO_MEMORY
0x98130021: S2S: No Memory

EC_E_S2SMBX_NO_DESCRIPTOR
0x98130022: S2S: No Descriptor

EC_E_S2SMBX_DEST_SLAVE_NOT_FOUND
0x98130023: S2S: Destination Slave not found

EC_E_MASTER_RED_STATE_INACTIVE
0x98130024: APP: Master Redundancy State is INACTIVE (e.g. API not allowed in current Master Redundancy State)

EC_E_MASTER_RED_STATE_ACTIVE
0x98130025: APP: Master Redundancy State is ACTIVE (e.g. API not allowed in current Master Redundancy State)

EC_E_JUNCTION_RED_LINE_BREAK
0x98130026: Junction redundancy line break

EC_E_VALIDATION_ERROR
0x98130027: Validation error (validation data mismatch)

EC_E_TIMEOUT_WAITING_FOR_DC
0x98130028: Timeout waiting for DC

EC_E_TIMEOUT_WAITING_FOR_DCM
0x98130029: Timeout waiting for DCM

EC_E_SIGNATURE_MISMATCH
0x98130030: Signature mismatch

EC_E_PDIWATCHDOG
0x98130031: PDI watchdog expired

EC_E_BAD_CONNECTION
0x98130032: Bad connection

EC_E_XML_INCONSISTENT
0x98130033: ENI: Inconsistent content

7.3 DCM Error Codes

DCM_E_ERROR
0x981201C0: Unspecific DCM Error

DCM_E_NOTINITIALIZED
0x981201C1: Not initialized

DCM_E_MAX_CTL_ERROR_EXCEED
0x981201C2: DCM controller - synchronization out of limit

DCM_E_NOMEMORY
0x981201C3: Not enough memory

DCM_E_INVALID_HWLAYER
0x981201C4: Hardware layer - (BSP) invalid

DCM_E_TIMER_MODIFY_ERROR
0x981201C5: Hardware layer - error modifying timer

DCM_E_TIMER_NOT_RUNNING
0x981201C6: Hardware layer - timer not running

DCM_E_WRONG_CPU
0x981201C7: Hardware layer - function called on wrong CPU

DCM_E_INVALID_SYNC_PERIOD
0x981201C8: Invalid DC sync period length (invalid clock master?)

DCM_E_INVALID_SETVAL
0x981201C9: DCM controller SetVal too small

DCM_E_DRIFT_TO_HIGH
0x981201CA: DCM controller - Drift between local timer and ref clock to high

DCM_E_BUS_CYCLE_WRONG
0x981201CB: DCM controller - Bus cycle time (dwBusCycleTimeUsec) doesn't match real cycle

DCX_E_NO_EXT_CLOCK
0x981201CC: DCX controller - No external synchronization clock found

DCM_E_INVALID_DATA
0x981201CD: DCM controller - Invalid data

7.4 ADS over EtherCAT® (AoE) Error Codes

EC_E_AOE_NO_RTIME
0x9813000D: AoE: No Rtime

EC_E_AOE_LOCKED_MEMORY
0x9813000E: AoE: Allocation locked memory

EC_E_AOE_MAILBOX
0x9813000F: AoE: Insert mailbox error

EC_E_AOE_WRONG_HMSG
0x98130010: AoE: Wrong receive HMSG

EC_E_AOE_BAD_TASK_ID
0x98130011: AoE: Bad task ID

EC_E_AOE_NO_IO
0x98130012: AoE: No IO

EC_E_AOE_UNKNOWN_AMS_COMMAND
0x98130013: AoE: Unknown ADS command

EC_E_AOE_WIN32
0x98130014: AoE: Win 32 error

EC_E_AOE_LOW_INSTALL_LEVEL
0x98130015: AoE: Low installation level

EC_E_AOE_NO_DEBUG
0x98130016: AoE: No debug available

EC_E_AOE_AMS_SYNC_WIN32
0x98130017: AoE: Sync Win 32 error

EC_E_AOE_AMS_SYNC_TIMEOUT
0x98130018: AoE: Sync Timeout

EC_E_AOE_AMS_SYNC_AMS
0x98130019: AoE: Sync AMS error

EC_E_AOE_AMS_SYNC_NO_INDEX_MAP
0x9813001A: AoE: Sync no index map

EC_E_AOE_TCP_SEND
0x9813001B: AoE: TCP send error

EC_E_AOE_HOST_UNREACHABLE
0x9813001C: AoE: Host unreachable

EC_E_AOE_INVALIDAMSFRAGMENT
0x9813001D: AoE: Invalid AMS fragment

EC_E_AOE_NO_LOCKED_MEMORY
0x9813001E: AoE: No allocation locked memory

EC_E_AOE_MAILBOX_FULL
0x9813001F: AoE: Mailbox full

7.5 CAN application protocol over EtherCAT® (CoE) SDO Error Codes

EC_E_SDO_ABORTCODE_TOGGLE
0x98110040: SLV: SDO: Toggle bit not alternated (CoE abort code 0x05030000 of slave)

EC_E_SDO_ABORTCODE_TIMEOUT
0x98110041: SLV: SDO: Protocol timed out (CoE abort code 0x05040000 of slave)

EC_E_SDO_ABORTCODE_CCS_SCS
0x98110042: SLV: SDO: Client/server command specifier not valid or unknown (CoE abort code 0x05040001 of slave)

EC_E_SDO_ABORTCODE_BLK_SIZE
0x98110043: SLV: SDO: Invalid block size (block mode only) (CoE abort code 0x05040002 of slave)

EC_E_SDO_ABORTCODE_SEQNO
0x98110044: SLV: SDO: Invalid sequence number (block mode only) (CoE abort code 0x05040003 of slave)

EC_E_SDO_ABORTCODE_CRC
0x98110045: SLV: SDO: CRC error (block mode only) (CoE abort code 0x05040004 of slave)

EC_E_SDO_ABORTCODE_MEMORY
0x98110046: SLV: SDO: Out of memory (CoE abort code 0x05040005 of slave)

EC_E_SDO_ABORTCODE_ACCESS
0x98110047: SLV: SDO: Unsupported access to an object (CoE abort code 0x06010000 of slave)

EC_E_SDO_ABORTCODE_WRITEONLY
0x98110048: SLV: SDO: Attempt to read a write only object (CoE abort code 0x06010001 of slave)

EC_E_SDO_ABORTCODE_READONLY
0x98110049: SLV: SDO: Attempt to write a read only object (CoE abort code 0x06010002 of slave)

EC_E_SDO_ABORTCODE_INDEX
0x9811004A: SLV: SDO: Object does not exist in the object dictionary (CoE abort code 0x06020000 of slave)

EC_E_SDO_ABORTCODE_PDO_MAP
0x9811004B: SLV: SDO: Object cannot be mapped to the PDO (CoE abort code 0x06040041 of slave)

EC_E_SDO_ABORTCODE_PDO_LEN
0x9811004C: SLV: SDO: The number and length of the objects to be mapped would exceed PDO length (CoE abort code 0x06040042 of slave)

EC_E_SDO_ABORTCODE_P_INCOMP
0x9811004D: SLV: SDO: General parameter incompatibility reason (CoE abort code 0x06040043 of slave)

EC_E_SDO_ABORTCODE_I_INCOMP
0x9811004E: SLV: SDO: General internal incompatibility in the device (CoE abort code 0x06040047 of slave)

EC_E_SDO_ABORTCODE_HARDWARE
0x9811004F: SLV: SDO: Access failed due to a hardware error (CoE abort code 0x06060000 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH
0x98110050: SLV: SDO: Data type does not match, length of service parameter does not match (CoE abort code 0x06070010 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_HIGH
0x98110051: SLV: SDO: Data type does not match, length of service parameter too high (CoE abort code 0x06070012 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_LOW
0x98110052: SLV: SDO: Data type does not match, length of service parameter too low (CoE abort code 0x06070013 of slave)

EC_E_SDO_ABORTCODE_OFFSET
0x98110053: SLV: SDO: Sub-index does not exist (CoE abort code 0x06090011 of slave)

EC_E_SDO_ABORTCODE_VALUE_RANGE
0x98110054: SLV: SDO: Value range of parameter exceeded (only for write access) (CoE abort code 0x06090030 of slave)

EC_E_SDO_ABORTCODE_VALUE_TOO_HIGH
0x98110055: SLV: SDO: Value of parameter written too high (CoE abort code 0x06090031 of slave)

EC_E_SDO_ABORTCODE_VALUE_TOO_LOW
0x98110056: SLV: SDO: Value of parameter written too low (CoE abort code 0x06090032 of slave)

EC_E_SDO_ABORTCODE_MINMAX
0x98110057: SLV: SDO: Maximum value is less than minimum value (CoE abort code 0x06090036 of slave)

EC_E_SDO_ABORTCODE_GENERAL
0x98110058: SLV: SDO: General error (CoE abort code 0x08000000 of slave)

EC_E_SDO_ABORTCODE_TRANSFER
0x98110059: SLV: SDO: Data cannot be transferred or stored to the application (CoE abort code 0x08000020 of slave)

EC_E_SDO_ABORTCODE_TRANSFER_LOCAL_CONTROL
0x9811005A: SLV: SDO: Data cannot be transferred or stored to the application because of local control (CoE abort code 0x08000021 of slave)

EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE
0x9811005B: SLV: SDO: Data cannot be transferred or stored to the application because of the present device state (CoE abort code 0x08000022 of slave)

EC_E_SDO_ABORTCODE_DICTIONARY
0x9811005C: SLV: SDO: 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) (CoE abort code 0x08000023 of slave)

EC_E_SDO_ABORTCODE_UNKNOWN
0x9811005D: SLV: SDO: Unknown code (Unknown CoE abort code of slave)

EC_E_SDO_ABORTCODE_MODULE_ID_LIST_NOT_MATCH
0x9811005E: Detected Module Ident List (0xF030) and Configured Module Ident list (0xF050) does not match

EC_E_SDO_ABORTCODE_SI_NOT_WRITTEN
0x98130004: SLV: SDO: Sub Index cannot be written, SI0 must be 0 for write access (CoE abort code 0x06010003 of slave)

EC_E_SDO_ABORTCODE_CA_TYPE_MISM
0x98130005: SLV: SDO: Complete access not supported for objects of variable length such as ENUM object types (CoE abort code 0x06010004 of slave)

EC_E_SDO_ABORTCODE_OBJ_TOO_BIG
0x98130006: SLV: SDO: Object length exceeds mailbox size (CoE abort code 0x06010005 of slave)

EC_E_SDO_ABORTCODE_PDO_MAPPED
0x98130007: SLV: SDO: Object mapped to RxPDO, SDO Download blocked (CoE abort code 0x06010006 of slave)

7.6 File Transfer over EtherCAT® (FoE) Error Codes

EC_E_FOE_ERRCODE_NOTDEFINED
0x98110060: SLV: ERROR FoE: not defined (FoE Error Code 0 (0x8000) of slave)

EC_E_FOE_ERRCODE_NOTFOUND
0x98110061: SLV: ERROR FoE: not found (FoE Error Code 1 (0x8001) of slave)

EC_E_FOE_ERRCODE_ACCESS
0x98110062: SLV: ERROR FoE: access denied (FoE Error Code 2 (0x8002) of slave)

EC_E_FOE_ERRCODE_DISKFULL
0x98110063: SLV: ERROR FoE: disk full (FoE Error Code 3 (0x8003) of slave)

EC_E_FOE_ERRCODE_ILLEGAL
0x98110064: SLV: ERROR FoE: illegal (FoE Error Code 4 (0x8004) of slave)

EC_E_FOE_ERRCODE_PACKENO
0x98110065: SLV: ERROR FoE: packet number wrong (FoE Error Code 5 (0x8005) of slave)

EC_E_FOE_ERRCODE_EXISTS
0x98110066: SLV: ERROR FoE: already exists (FoE Error Code 6 (0x8006) of slave)

EC_E_FOE_ERRCODE_NOUSER
0x98110067: SLV: ERROR FoE: no user (FoE Error Code 7 (0x8007) of slave)

EC_E_FOE_ERRCODE_BOOTSTRAPONLY
0x98110068: SLV: ERROR FoE: bootstrap only (FoE Error Code 8 (0x8008) of slave)

EC_E_FOE_ERRCODE_NOTINBOOTSTRAP
0x98110069: SLV: ERROR FoE: Downloaded file name is not valid in Bootstrap state (FoE Error Code 9 (0x8009) of slave)

EC_E_FOE_ERRCODE_INVALIDPASSWORD
0x9811006A: SLV: ERROR FoE: no rights (FoE Error Code 10 (0x800A) of slave)

EC_E_FOE_ERRCODE_PROGERROR
0x9811006B: SLV: ERROR FoE: program error (FoE Error Code 11 (0x800B) of slave)

EC_E_FOE_ERRCODE_INVALID_CHECKSUM
0x9811006C: FoE: Wrong checksum

EC_E_FOE_ERRCODE_INVALID_FIRMWARE
0x9811006D: SLV: ERROR FoE: Firmware does not fit for Hardware (FoE Error Code 13 (0x800D) of slave)

EC_E_FOE_ERRCODE_NO_FILE
0x9811006F: SLV: ERROR FoE: No file to read (FoE Error Code 15 (0x800F) of slave)

EC_E_NO_FOE_SUPPORT_BS
0x9811010F: APP: ERROR FoE: Protocol not supported in boot strap (e.g. Application requested FoE in Bootstrap although slave does not support this)

EC_E_FOE_ERRCODE_MAX_FILE_SIZE
0x9811017A: APP: ERROR FoE: File is bigger than max file size (e.g. Slave returned more data than the buffer provided by application can store.)

EC_E_FOE_ERRCODE_FILE_HEAD_MISSING
0x98130001: SLV: ERROR FoE: File header does not exist (FoE Error Code 16 (0x8010) of slave)

EC_E_FOE_ERRCODE_FLASH_PROBLEM
0x98130002: SLV: ERROR FoE: Flash problem (FoE Error Code 17 (0x8011) of slave)

EC_E_FOE_ERRCODE_FILE_INCOMPATIBLE
0x98130003: SLV: ERROR FoE: File incompatible (FoE Error Code 18 (0x8012) of slave)

7.7 Servo Drive Profil over EtherCAT® (SoE) Error Codes

EC_E_SOE_ERRORCODE_INVALID_ACCESS
0x98110078: ERROR SoE: Invalid access to element 0

EC_E_SOE_ERRORCODE_NOT_EXIST
0x98110079: ERROR SoE: Does not exist

EC_E_SOE_ERRORCODE_INVL_ACC_ELEM1
0x9811007A: ERROR SoE: Invalid access to element 1

EC_E_SOE_ERRORCODE_NAME_NOT_EXIST
0x9811007B: ERROR SoE: Name does not exist

EC_E_SOE_ERRORCODE_NAME_UNDERSIZE
0x9811007C: ERROR SoE: Name undersize in transmission

EC_E_SOE_ERRORCODE_NAME_OVERSIZE
0x9811007D: ERROR SoE: Name oversize in transmission

EC_E_SOE_ERRORCODE_NAME_UNCHANGE
0x9811007E: ERROR SoE: Name unchangeable

EC_E_SOE_ERRORCODE_NAME_WR_PROT
0x9811007F: ERROR SoE: Name currently write-protected

EC_E_SOE_ERRORCODE_UNDERS_TRANS
0x98110080: ERROR SoE: Attribute undersize in transmission

EC_E_SOE_ERRORCODE_OVERS_TRANS
0x98110081: ERROR SoE: Attribute oversize in transmission

EC_E_SOE_ERRORCODE_ATTR_UNCHANGE
0x98110082: ERROR SoE: Attribute unchangeable

EC_E_SOE_ERRORCODE_ATTR_WR_PROT
0x98110083: ERROR SoE: Attribute currently write-protected

EC_E_SOE_ERRORCODE_UNIT_NOT_EXIST
0x98110084: ERROR SoE: Unit does not exist

EC_E_SOE_ERRORCODE_UNIT_UNDERSIZE
0x98110085: ERROR SoE: Unit undersize in transmission

EC_E_SOE_ERRORCODE_UNIT_OVERSIZE
0x98110086: ERROR SoE: Unit oversize in transmission

EC_E_SOE_ERRORCODE_UNIT_UNCHANGE
0x98110087: ERROR SoE: Unit unchangeable

EC_E_SOE_ERRORCODE_UNIT_WR_PROT
0x98110088: ERROR SoE: Unit currently write-protected

EC_E_SOE_ERRORCODE_MIN_NOT_EXIST
0x98110089: ERROR SoE: Minimum input value does not exist

EC_E_SOE_ERRORCODE_MIN_UNDERSIZE
0x9811008A: ERROR SoE: Minimum input value undersize in transmission

EC_E_SOE_ERRORCODE_MIN_OVERSIZE
0x9811008B: ERROR SoE: Minimum input value oversize in transmission

EC_E_SOE_ERRORCODE_MIN_UNCHANGE
0x9811008C: ERROR SoE: Minimum input value unchangeable

EC_E_SOE_ERRORCODE_MIN_WR_PROT
0x9811008D: ERROR SoE: Minimum input value currently write-protected

EC_E_SOE_ERRORCODE_MAX_NOT_EXIST
0x9811008E: ERROR SoE: Maximum input value does not exist

EC_E_SOE_ERRORCODE_MAX_UNDERSIZE
0x9811008F: ERROR SoE: Maximum input value undersize in transmission

EC_E_SOE_ERRORCODE_MAX_OVERSIZE
0x98110090: ERROR SoE: Maximum input value oversize in transmission

EC_E_SOE_ERRORCODE_MAX_UNCHANGE
0x98110091: ERROR SoE: Maximum input value unchangeable

EC_E_SOE_ERRORCODE_MAX_WR_PROT
0x98110092: ERROR SoE: Maximum input value currently write-protected

EC_E_SOE_ERRORCODE_DATA_NOT_EXIST
0x98110093: ERROR SoE: Data item does not exist

EC_E_SOE_ERRORCODE_DATA_UNDERSIZE
0x98110094: ERROR SoE: Data item undersize in transmission

EC_E_SOE_ERRORCODE_DATA_OVERSIZE
0x98110095: ERROR SoE: Data item oversize in transmission

EC_E_SOE_ERRORCODE_DATA_UNCHANGE
0x98110096: ERROR SoE: Data item unchangeable

EC_E_SOE_ERRORCODE_DATA_WR_PROT
0x98110097: ERROR SoE: Data item currently write-protected

EC_E_SOE_ERRORCODE_DATA_MIN_LIMIT
0x98110098: ERROR SoE: Data item less than minimum input value limit

EC_E_SOE_ERRORCODE_DATA_MAX_LIMIT
0x98110099: ERROR SoE: Data item exceeds maximum input value limit

EC_E_SOE_ERRORCODE_DATA_INCOR
0x9811009A: ERROR SoE: Data item incorrect

EC_E_SOE_ERRORCODE_PASWD_PROT
0x9811009B: ERROR SoE: Data item protected by password

EC_E_SOE_ERRORCODE_TEMP_UNCHANGE
0x9811009C: ERROR SoE: Data item temporary unchangeable (in AT or MDT)

EC_E_SOE_ERRORCODE_INVL_INDIRECT
0x9811009D: ERROR SoE: Invalid indirect

EC_E_SOE_ERRORCODE_TEMP_UNCHANGE1
0x9811009E: ERROR SoE: Data item temporary unchangeable (parameter or opmode)

EC_E_SOE_ERRORCODE_ALREADY_ACTIVE
0x9811009F: ERROR SoE: Command already active

EC_E_SOE_ERRORCODE_NOT_INTERRUPT
0x98110100: ERROR SoE: Command not interruptible

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL
0x98110101: ERROR SoE: Command not available (in this phase)

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL1
0x98110102: ERROR SoE: Command not available (invalid parameter)

EC_E_SOE_ERRORCODE_DRIVE_NO
0x98110103: ERROR SoE: Response drive number not identical with requested drive number

EC_E_SOE_ERRORCODE_IDN
0x98110104: ERROR SoE: Response IDN not identical with requested IDN

EC_E_SOE_ERRORCODE_FRAGMENT_LOST
0x98110105: ERROR SoE: At least one fragment lost

EC_E_SOE_ERRORCODE_BUFFER_FULL
0x98110106: ERROR SoE: RX buffer full (EtherCAT call with to small data-buffer)

EC_E_SOE_ERRORCODE_NO_DATA
0x98110107: ERROR SoE: No data state

EC_E_SOE_ERRORCODE_NO_DEFAULT_VALUE
0x98110108: ERROR SoE: No default value

EC_E_SOE_ERRORCODE_DEFAULT_LONG
0x98110109: ERROR SoE: Default value transmission too long

EC_E_SOE_ERRORCODE_DEFAULT_WP
0x9811010A: ERROR SoE: Default value cannot be changed, read only

EC_E_SOE_ERRORCODE_INVL_DRIVE_NO
0x9811010B: ERROR SoE: Invalid drive number

EC_E_SOE_ERRORCODE_GENERAL_ERROR
0x9811010C: ERROR SoE: General error

EC_E_SOE_ERRCODE_NO_ELEM_ADR
0x9811010D: ERROR SoE: No element addressed

7.8 Remote API Error Codes

EC_E_SOCKET_DISCONNECTED
0x9811017D: RAS: Socket disconnected (e.g. IP connection terminated or lost)

EMRAS_E_INVALIDCOOKIE
0x98110181: RAS: Invalid Cookie (e.g.obsolete)

EMRAS_E_MULSRVDISMULCON
0x98110183: RAS: Connect 2nd server denied because Multi Server support is disabled (obsolete)

EMRAS_E_LOGONCANCELLED
0x98110184: RAS: Logon canceled (Server-side connection reject while opening a client connection.)

EMRAS_E_INVALIDVERSION
0x98110186: RAS: Invalid Version (Connection reject because of using mismatching protocol versions on client and server side)

EMRAS_E_INVALIDACCESSCONFIG
0x98110187: RAS: Access configuration is invalid (e.g. SPoC access configuration invalid)

EMRAS_E_ACCESSLESS
0x98110188: RAS: No access to this call at this access level (e.g. a higher SPoC access level is needed to use the called Remote API function)

EMRAS_E_INVALIDDATARECEIVED
0x98110189: RAS: Invalid data received (communication corrupted)

EMRAS_EVT_SERVERSTOPPED
0x98110191: RAS: Server stopped (e.g. connection dropped because of Remote API Server stop)

EMRAS_EVT_WDEXPIRED
0x98110192: RAS: Watchdog expired (e.g. connection dropped because of missing keep-alive messages)

EMRAS_EVT_RECONEXPIRED
0x98110193: RAS: Reconnect expired (obsolete)

EMRAS_EVT_CLIENTLOGON
0x98110194: RAS Server: Client logged on

EMRAS_EVT_RECONNECT
0x98110195: RAS: obsolete

EMRAS_EVT_SOCKCHANGE
0x98110196: RAS: Socket exchanged after reconnect (obsolete)

EMRAS_EVT_CLNTDISC
0x98110197: RAS: Client disconnect

EMRAS_E_ACCESS_NOT_FOUND
0x98110198: RAS: Access not configured for this call (e.g. SPoC access configuration missing)

EMRAS_E_TOKEN_MISSING
0x98110199: RAS: Token missing

EMRAS_E_TOKEN_INVALID
0x9811019A: RAS: Token invalid

EMRAS_E_TOKEN_DENIED
0x9811019B: RAS: Token denied

EMRAS_E_TLS_CERTIFICATE_ERROR
0x9811019C: RAS: TLS certificate error

EMRAS_E_TLS_PRIVATEKEY_ERROR
0x9811019D: RAS: TLS private key error

EMRAS_E_TLS_HANDSHAKE_FAILED
0x9811019E: RAS: TLS handshake failed